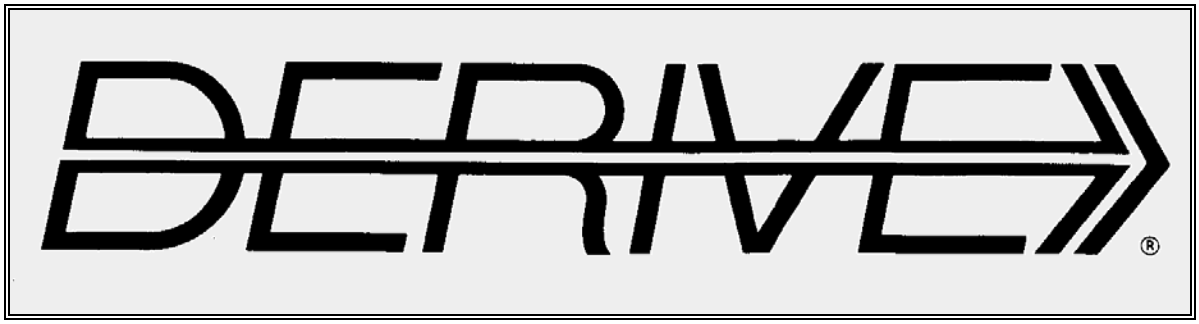
THE DERIVE - NEWSLETTER #55

THE BULLETIN OF THE



USER GROUP

+ CAS-TI

Contents:

September 2004

Derive User Group Meeting 2004

We used the occasion of TIME 2004 in Montréal to have our User Group Meeting. We had an intensive exchange of information and proposals between the program developers (Albert Rich & Theresa Shelby), the sellers (Bernhard Kutzler & Lana Moore) and the members of our group.
In the next DNL I´ll give an extended report of our meeting.

Cvetka Rojko, Clouds in Fort Chambly, Montréal

Recommended Websites:

Marlene Torres-Skoumal´s website: www.cas-time.com

Fernando Sivit's site: www.fermatsi.org

Dear DUG Members,

First of all I have to apologize for being late with this issue of our newsletter. My excuse is that I had so many duties for CAS-related tasks in the last weeks. But fortunately all of them are coming to an end. I am happy to announce that the TIME 2004-Conference CD is ready and will be available very soon (bk-teachware). TIME 2004 was one more highlight in the long row of our DERIVE/TI and ACDCA-Conferences. Michel Beaudin and his team did a perfect job, all our congratulations and thanks to all of you.

In the very next time you can download the revised version of DNL#4 (Dec. 1991). In this newsletter I announced the first Derive Spring School in Krems, Lower Austria for March 1992. And this was the conference where all has begun. All of us who participated remember the "Spirit of Krems" and we are still inspired by the atmosphere of our first meeting and became close friends for years. And we were moved again by the "Spirit of Krems" in Canada this summer – although some of us (only the male, of course) – have more grey hairs, ...

I hope that I can offer a striking mixture of contributions in this issue: Frank and Bert´s paper demonstrate that ideas born years ago are still demanding using the latest technology, Gerhard Hagen presents an easy to understand first introduction in FUZZY Logic, Don Phillips provides the second part of his Financial Mathematics for Derive (- a competition between him and me!), and finally Tania Koller gives once more an excellent example how to transfer educational ideas from one platform to the other. Many thanks to all of you. Not to forget our Mr. Titbits, who after Stephan Welke also accepted the challenge of Rüdeger Baumann´s "Josephus Problem".

Johann proposed to publish problems for our members on our website monthly. So let´s have a test and wait how many of you will treat Steven´s problem which you can find on the last page.

With my best regards

Josef

The *DERIVE-NEWSLETTER* is the Bulletin of the *DERIVE* & CAS-*TI User Group*. It is published at least four times a year. The goals of the *DNL* are to enable the exchange of experiences made with *DERIVE* and the *TI*-89/92/*Voyage* 200 as well as to create a group to discuss the possibilities of new methodical and didactical manners in teaching mathematics.

As many of the *DERIVE* Users are also using the CAS-*TIs* the *DNL* tries to combine the applications of these modern technologies.

Contributions:
Please send all contributions to the Editor. Non-English speakers are encouraged to write their contributions in English to reinforce the international touch of the *DNL*. It must be said, though, that non-English articles will be warmly welcomed nonetheless. Your contributions will be edited but not assessed. By submitting articles the author gives his consent for reprinting it in the *DNL*. The more contributions you will send, the more lively and richer in contents the *DERIVE* & CAS-*TI Newsletter* will be.

Editor: Mag. Josef Böhm
A-3042 Würmla
D´Lust 1
Austria
Phone/FAX: 43-(0)2275/8207
e-mail: nojo.boehm@pgv.at

Next issue: December 2004
Deadline: 15 November 2004

Preview: Contributions waiting to be published

Finite continued fractions St. Welke, GER
Some simulations of Random Experiments, J. Böhm, AUT
Wonderful World of Pedal Curves, J. Böhm
Another Task for End Examination, J. Lechner, AUT
Tools for 3D-Problems, P. Lüke-Rosendahl, GER
ANOVA with *DERIVE* & *TI*, M. R. Phillips, USA
Hill-Encription, J. Böhm
CAD-Design with *DERIVE* and the *TI*, J. Böhm
Avoiding Convolution and Transforming Methods, M. Lesmes-Acosta, COL
Farey Sequences on the *TI,* M. Lesmes-Acosta, COL
Simulating a Graphing Calculator in *DERIVE*, J. Böhm, AUT
Henon & Co, J. Böhm
Pringles, B. Grabinger, GER
Quadratic Approximation for Integration, G. Mann & N. Zehavi, ISR
Quadratic Optimization & Challenges from Fermat, Bj. Felsager, DEN
Applications of Moore Penrose Inverses, K. Schmidt, GER
and
Setif, FRA; Vermeylen, BEL; Leinbach, USA; Koller, AUT,
Keunecke, GER, Alm, SWE.....and others

Impressum:
Medieninhaber: *DERIVE* User Group, A-3042 Würmla, D´Lust 1, AUSTRIA
Richtung: Fachzeitschrift
Herausgeber: Mag.Josef Böhm
Herstellung: Selbstverlag

Discussion about sign(0)

In May and June we had a very emotional discussion about the TI´s and Derive´s interpretation of `sign(0)`.

The first message delivered by an Austrian user (forwarded by Kim Hendrickx, TI-Europe) was:

"The sign-function is programmed false, sign(0) = 0 and not ±1 which is given by the Voyage 200"

This remark was confirmed by a German user. He says that

"the definition of sign(x) = x / |x| is valid world wide either for real or for complex numbers. For sign(0) doesn´t exist a common definition, so sign(0) is considered undefined. For complex numbers sign(0) is uniformly undefined and as the real numbers are a subset of the complex numbers it seems to be reasonable to accept this also for real numbers.

Often for real numbers sign(0) is defined as 0 which is practicable for many applications – even Mathematica 5 – does it. TI has programmed an unacceptable definition, because sign(0) is not well-defined. Or is there a special motivation for TI to define sign(0) as ±1?"

My answer was

Dear Kim,

sign(0) is obviously a "question of belief". The TI is programmed like Derive. Among the real numbers sign(0) is ±1, among complex numbers sign(z) simplifies to the point on the unit circle in the complex plane that has the same phase angle as z . As 0 has no phase angle Derive returns also ±1.

I am convinced of the mathematical competence of Albert Rich and David Stoutemyer and I follow their definition. The fact that Mathematica does this in another way is no "sufficient condition" that sign(0) = 0.

To overcome problems with the jump for x = 0 one can redefine sign(0) = 0 to make it well-defined for certain special needs.

This was not a "good" answer, because a very emotional mail came back.

"sign(0) seems to be a "question of belief". This sentence irritates my mind. Mathematics is not a question of belief. I would like to compare this with our traffic rules:

If a driver in England will drive on the right side because he is believing that he has to drive right, he will not get far. One has to know that by definition it is a must to drive at the left in England and to drive at the right in Germany.

And it is the same with the sign-function. In all books I know I can find:

the right sided limit of sign(x) with x tending to 0 is +1 and
the left sided limit of sign(x) with x tending to 0 is –1.
sign(0) = 0 or sign(0) is not defined.

I am only talking about the real numbers, for complex numbers we have other rules.

I repeat that we had to change our math education because a multivalued assigment is not a function. And the concept of a function cannot be changed. I´d like to explain this with another example:

Driving a curve is a unique function of movement of the steering wheel:

right steering angle→ right-hand bend, left steering angle → left-hand bend
steering angle 0 → we cannot let the car go anywhere!!

Turning on the headlight is a multivalued assigment:

2. switch full beam → full beam, 2. switch passing beam → passing beam.

I wrote to Albert Rich:

Dear Albert,

just recently I received some requests why Derive returns ±1 for sign(0) in contrary to most of the textbooks and other CAS-programs which return sign(0)=0. We know that it makes a difference if working with real or complex numbers.

Teachers - and students are sometimes confused about ±1.

Many thanks for your answer in advance.

Reliable as ever Albert answered immediately:

Hello Josef,

Good question. If Derive were just a numerical calculator, SIGN(0) could be arbitrarily assigned. I believe some of Derive's competitors allow users to make that assignment. However, since Derive is a symbolic math system that at least attempts to be internally consistent, the value of SIGN(0) can not be so arbitrarily assigned.

In order to simplify mathematical expressions involving not only SIGN but also the much more commonly used function ABS, the most powerful and generally applicable transformation rules possible must be available for use.

The three obvious choices for the value of SIGN(0) are 0, 1, and ±1 (plus-or-minus one). Derive needs to be able to simplify expressions of the form

SIGN(u)^(2*n)

to 1, if u is real-valued expression and n is an integer. This rule would be invalid if SIGN(0) simplified to 0. In Derive, ABS(u) is transformed internally to u/SIGN(u) and then transformed back to ABS(u) for output. If SIGN(0) simplified to 0, then ABS(0) would be transformed to 0/0 which is indeterminate. So 0 is out.

On the other hand, if SIGN(0) simplified to 1, the above two problems would be resolved but the very essential rule SIGN(-u) simplifies to -SIGN(u) would be invalid when u=0. So 1 is out.

Happily, assigning SIGN(0) the value ±1 (plus-or-minus one) resolves all these problems, and really captures the essence of what SIGN of 0 should mean. Think of it as a real-valued quantity whose magnitude is 1, but whose sign is unknown.

Analogously, the Derive "constant" for an arbitrary point on the unit circle named unit_circle is a complex-valued quantity whose magnitude is 1, but whose phase angle is unknown. Note that `unit_circle` is the result Derive gives as the solution of ABS(z) = 1.

I do not think ±1 (plus-or-minus one) is really all that mysterious a beast. After all, it appears front and center as the coefficient of the radical term in the solution of the quadratic equation. Also it is tantalizingly like the indeterminacy that underlies quantum physics... Anyway, I hope this long-winded explanation helps.

Aloha,
Albert D. Rich
Co-author of Derive

Both colleagues which raised the question were not satisfied. I collected their concerns and wrote another message to Albert Rich:

Dear Albert,
the sign(0) discussion is very interresting. I sent your comments to Prof. Paditz and Mr. Siedler. Prof. Paditz claims that the two-valued arithmetic causes some inconsistencies - see below.

So for example: sign0)+sign(0) = undef, but 2*sign(0) = ± 2 ....

You can see other examples below. I translate Prof. Paditz´s mail. My comments are in blue.

1. expand((a + sign(0)*b)^2) = **a^2 ± 2*a*b + b^2** would be correct .
   but TI-89/voy200 (expand((a + sign(0)*b)^2) = **a^2 ± 2*a*b ± b^2**
   That´s only valid for the TIs, Derive returns a "?" The other results are the same in Derive

2. sign(0) - sign(0) = **0** would be correct .
   Derive: sign(0) - sign(0) = **undef**

3. (sign(0)) * (-sign(0)) = **-1** would be correct
   Derive: sign(0) * (-sign(0)) = **±1**

4. (sign(0)) * (sign(0)) = **1** would be correct
   Derive: (sign(0)) * (-sign(0)) = **±1**
   but (sign(0))^2 = **1**

5. (sign(0)) + (sign(0)) = **±2** would be correct
   Derive: (sign(0)) + (sign(0)) = **undef**
   but (sign(0))*2 = **±2**

Albert Rich again:

It is important to note that the value of each SIGN(0) (that is plus-or-minus one) in an expression is independent of the other occurrences of SIGN(0). Therefore, Derive simplifies

SIGN(0) + SIGN(0)

to ? since the value could be 2, 0, or -2 and Derive has no way of representing three value objects. Similar remarks apply to the other examples given below.

This was followed by another mail containing some objections to Albert´s arguments. **What is your opinion?**

MacDonald Phillips phillipsm@GAO.GOV

I just got a USB flash memory device (256 MB) and was wondering if I could run Derive 6 from it. The answer is yes. So, if you can't take your computer with you but will have access to a computer where ever you're going, you can take Derive and all your files along with you.
All you have to do is run the install font program from the Control Panel and install the Derive fonts first.

I installed my entire Derive 6 directory onto the USB flash memory and I can now run my copy of Derive with all my files on any compatable computer.

Regards,
Don

"There are only 10 types of people in the world: Those who understand binary and those who don't."

Marlene Torres-Skoumal marlenes@aon.at

Hello (CAS) friends,

I would greatly appreciate your having a look at my new website,

www.cas-time.com

and especially at the menu CAS FRIENDS where I have provided links to your websites. Any feedback you can offer for my next update to improve the quality of what we're collectively offering the educational world would be much appreciated.

Lynda Ball lball@unimelb.edu.au

Dear CAS-CAT interest group,

You may be interested to visit our newly updated CAS-CAT website. The CAS-CAT website now incorporates all of the technology research that we are doing in the Department of Science and Mathematics Education at the University of Melbourne as well as infomration about the research project that we affectionately referred to as "The CAS-CAT project".
The website will be updated regularly with our current research on CAS and other technologies. If you haven't yet visited the RITEMATHS website then you might want to follow the link to this site from CAS-CAT.

CAS mini conference

We are running a mini conference on CAS from 24-26 November, 2004. You will be able to access details directly from the CAS-CAT website in the next day or so, or else you can email me for further details.

Looking forward to catching up with many of you soon and hearing about your work with CAS.
Kind regards,
Lynda

Department of Science and Mathematics Education
University of Melbourne
Victoria 3010

CAS CAT Project: www.edfac.unimelb.edu.au/DSME/CAS-CAT/

Complex Zeros Graphically. The Modulus Surface

Franklin Demana and Bert Waits

There is an important and well-known connection between real zeros of functions of a single variable x and the x-intercepts of the graph of the function. That is, r is a real solution to the equation $f(x) = 0$ if and only if r is an x-intercept of the graph of f. This is a natural geometric connection. Most teachers are not aware of a geometric connection for the complex solutions to the equation $f(x) = 0$. In this article we will show there is a very useful geometric connection between the complex zeros of a function and points on a 3-dimensional graph.

First, we introduce the *modulus surface* of a function f of a single variable. It turns out that any function of one variable can be used to define a surface in 3-space that provides an important and useful *geometric representation* of the complex zeros of the function.

Recall that the *absolute value* or *modulus* $|u|$ of the complex number $u = a + bi$ is defined to be the real number $|u| = \sqrt{a^2 + b^2}$. Let f be any function of one variable. The **modulus surface** of f is the surface defined by $z = g(x, y) = |f(u)|$, where u is the complex number $x + yi$. The function g is called the **modulus function** of f.

The value of z in $z = |f(u)|$ is a non-negative *real number* for every complex number $x + yi$. If we interpret the xy-plane as the complex plane, then we can think of $z = |f(u)|$ as a function of the two variables x and y where $u = x + yi$ (Figure 1). The point $(x, y, 0)$ corresponds to to the complex number $x + yi$. The horizontal axis of the xy-plane is the real axis and the vertical axis is the imaginary axis.

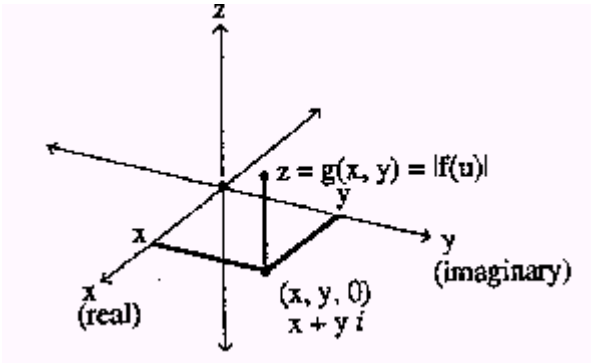


Figure 1

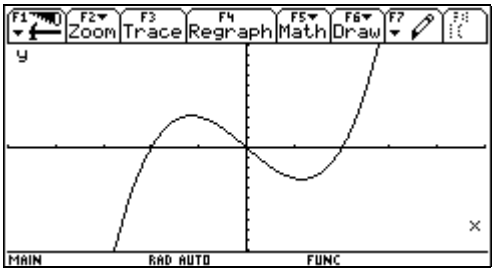


Figure 2 $f(x) = x^3 - 4x$
[-5,5] by [-10,10]

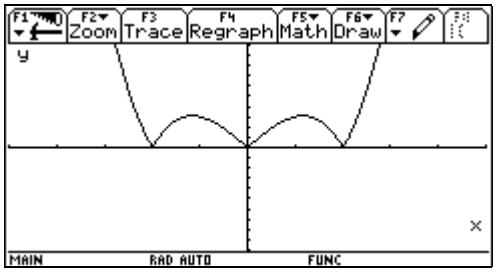


Figure 3
$z = |f(x)| = |x^3 - 4x|$

The trace of $z = |f(u)|$ in the xz-plane ($y = 0$) is the graph of $z = |f(x + 0 \cdot i)| = |f(x)|$. Let $f(x) = x^3 - 4x$. The graph of f in the viewing rectangle determined by $-5 \le x \le 5$ and $-10 \le y \le 10$ (denoted by [–5,5] by [–10,10]) is given in Figure 2. The trace of $z = |f(u)|$ in the xz-plane is the graph of $z = |x^3 - 4x|$ (Figure 3). The real zeros of f show up as the x-intercepts of the trace of the modulus function $z = |f(u)|$ in the xz-plane. These are points on the x-axis where the modulus

function $z = |f(u)|$ of $f(x) = x^3 - 4x$ touches the *xy*-plane. This observation is a special case of the more general result.

Theorem. Let $u = x + yi$ and let f be a function with domain and range that are subsets of the complex numbers. The complex number $a + bi$ is a zero of f if and only if the modulus surface $z = g(x, y) = |f(u)|$ touches the complex plane at $a + bi$. That is, $a + bi$ is a zero of f if and only if $(a, b, 0)$ is a point on the graph of $z = g(x, y) = |f(u)|$.

Proof. Assume $a + bi$ is a zero of f. Then $f(a + bi) = 0$. It follows that $z = g(a,b) = |f(a + bi)| = 0$. Thus, the modulus surface $z = g(x, y) = |f(u)|$ touches the *xy*-plane at $(a, b, 0)$, or at $a + bi$ when the *xy*-plane is considered the complex plane. Notice that $(a, b, 0)$ is a point on the graph of $z = g(x, y) = |f(u)|$ in this case. Moreover, the modulus surface never goes through the *xy*-plane (complex plane) because $z = |f(u)| \geq 0$ for all x and y.

Now, assume the modulus surface $z = g(x, y) = |f(u)|$ touches the complex plane at $a + bi$. That is, assume that $(a, b, 0)$ is a point of the graph of $z = |f(u)|$. This means that $z = 0$ at $u = a + bi$. Thus, $|f(a + bi)| = 0$. It follows that $f(a + bi) = 0$ because the modulus of a complex number is zero if and only if the complex number is zero.

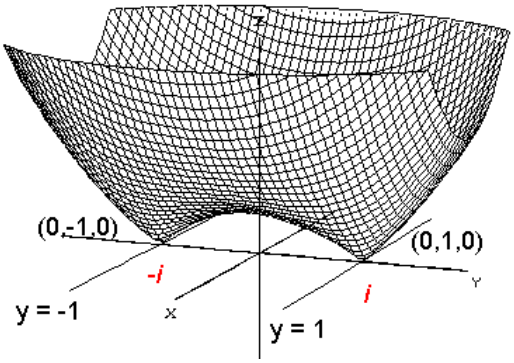
The key to apply this theorem to visualize the complex zeros of a function is the ability to easily graph a function of two variables. We now determine a geometric representation of the zeros of $f(x) = x^2 + 1$ – that is, draw a graph that shows all points where the modulus surface of f touches the *xy*-plane.

The modulus surface of f is the graph of $z = g(x, y) = |f(u)|$ where $u = x + yi$.

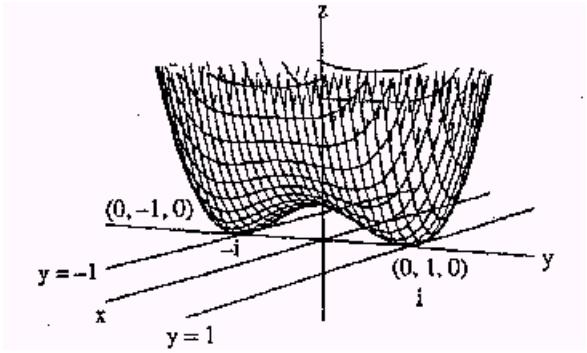
$$\begin{aligned} z &= |f(u)| \\ &= |f(x + yi)| \\ &= |(x + yi)^2 + 1| \\ &= |x^2 + 2xyi + y^2 i^2 + 1| \\ &= |x^2 - y^2 + 1 + 2xyi| \\ &= \sqrt{(x^2 - y^2 + 1)^2 + (2xy)^2} \qquad \text{(Why?)} \end{aligned}$$

A graph of the modulus surface of f in the viewing box determined by $-2 \leq x \leq 2$, $-2 \leq y \leq 2$, and $-5 \leq z \leq 5$ (denoted by $[-2,2] \times [-2,2] \times [-5,5]$ is given in Figure 4. The surface appears to touch the *xy*-plane at (0, 1, 0) and (0, –1, 0). We have added the coordinate axes and the lines $y = 1$ and $y = -1$ in the *xy*-plane to the graph in Figure 4 to help locate the coordinates of these points. *Master Grapher* [Waits and Demana, 1989] will overlay lines and the coordinate axes on a graph. Of course, we know that $f(x) = x^2 + 1$ has exactly two zeros, namely $\pm i$. Thus Figure 4 provides a **geometric representation** of the zeros of $f(x) = x^2 + 1$. Notice the zeros are the minimum points of the modulus surface.

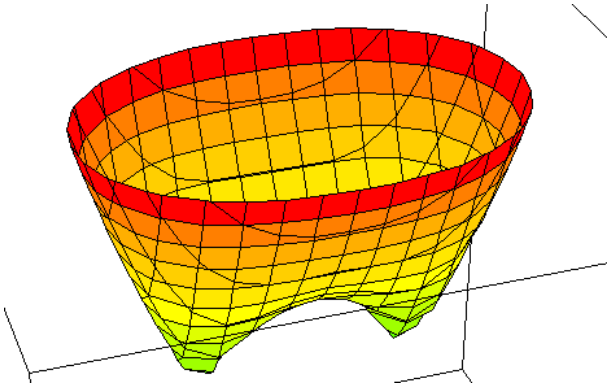
Now we *can see* the complex zeros!



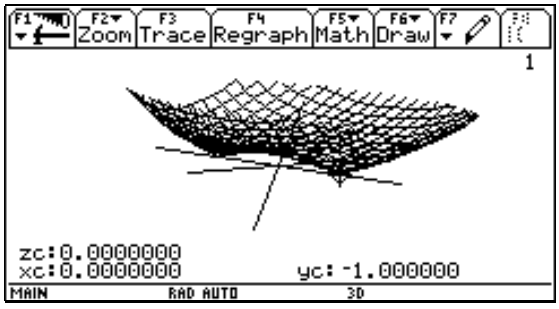
Derive



Master Grapher



DPGraph



Voyage 200

Figure 4, $z = \sqrt{(x^2 - y^2 + 1)^2 + 4x^2y^2}$

The graphs of $x = a$ and $y = b$ in 3-space are planes. However, in this article, when we refer to the lines $x = a$ and $y = b$ we mean the lines $x = a$ *and* $y = b$ in the *xy*-plane.

Can you predict what the geometric representation of the zeros of $f(x) = x^2 - 1$ will look like? You will find that the modulus surface of this function is the surface of the previous example rotated 90° about the *z*-axis. (Why?) The two minimum points where the modulus surface of $f(x) = x^2 - 1$ touches the *xy*-plane are on the *x*-axis. Of course, this means the zeros of *f* are real.

Next we determine a geometric representation of the complex zeros of $f(x) = x^2 + x + 1$.

Using the quadratic formula, the two zeros of *f* are $\frac{-1 \pm i\sqrt{3}}{2}$ or –0.5 ± 0.87*i*. First we determine the modulus function $z = g(x, y) = |f(u)|$ of *f*.

$$\begin{aligned} z &= |f(u)| \\ &= |f(x + yi)| \\ &= |(x + yi)^2 + (x + yi) + 1| \\ &= |(x^2 - y^2 + x + 1) + (2xy + y)i| \\ &= \sqrt{(x^2 - y^2 + x + 1)^2 + (2xy + y)^2} \end{aligned}$$

This should be a very welcome occasion to use a CAS!!

```
mod_surf(f) := |SUBST(f, x, x + y·i)|

                    2
mod_surf(x  + x + 1)

   4      3    2     2              2          4    2
√(x  + 2·x  + x ·(2·y  + 3) + 2·x·(y  + 1) + y  - y  + 1)
```

A geometric representation of the two complex zeros of f is given by the graph of $z = \sqrt{(x^2 - y^2 + x + 1)^2 + (2xy + y)^2}$, the modulus surface of f (Figure 5). We have added the coordinate axes and the lines $x = -0.5$, $y = 0.87$ and $y = -0.87$ to help locate the zeros from the graph. This graph confirms the zeros we determined algebraically using the quadratic formula. Again, notice the zeros are the minimum points of the modulus surface and we *can see* these complex zeros.

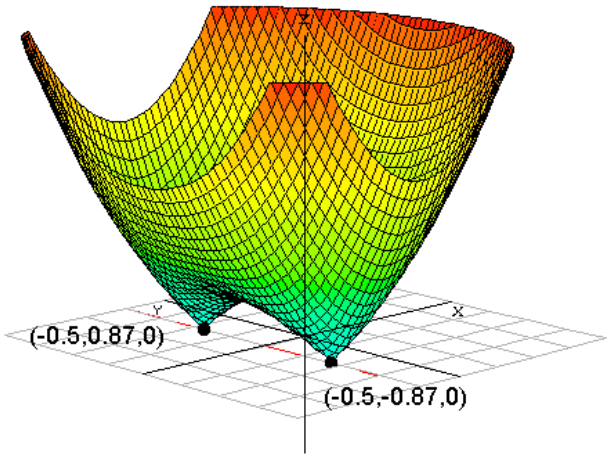


Figure 5

As our last example, we determine a geometric representation of the zeros of $f(x) = x^3 - x^2 + x - 1$. We will use the graph to estimate the zeros of *f*. This polynomial of degree three always has at least one real zero. It turns out that *f* also has two nonreal complex zeros. To obtain a geometric representation, we determine the modulus function of *f*. Here a symbolic manipulator would be useful.

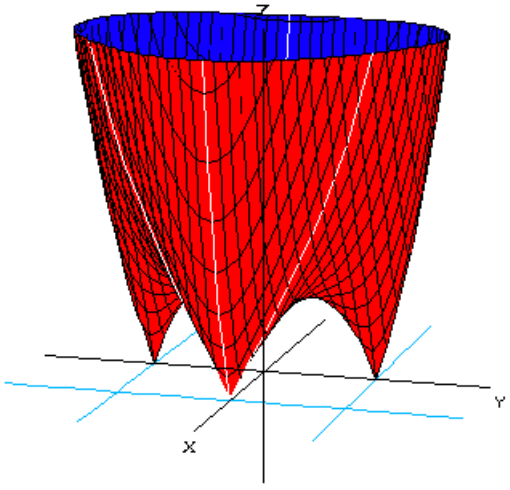


Figure 6 , $z = \sqrt{(x^3 - 3xy^2 - x^2 + y^2 + x - 1)^2 + (3x^2y - y^3 - 2xy + y)^2}$

A geometric representation of the zeros of *f* is given by the graph of

$$z = \sqrt{(x^3 - 3xy^2 - x^2 + y^2 + x - 1)^2 + (3x^2y - y^3 - 2xy + y)^2} \ ,$$

the modulus surface of f (Figure 6 and *Derive*-calculation below). It appears that the modulus surface touches the *xy*-plane at two points on the *y*-axis and on one point on the *x*-axis. By using zoom-in (or tracing) we can estimate these points to be (1,0,0), (0,1,0) and (0,–1,0). We have added the coordinate axes and the lines $x = 1$, $y = 1$ and $y = -1$ to help establish these estimates. These points correspond to the complex numbers 1, *i* and –*i* which are good estimates for the zeros of $f(x) = x^3 - x^2 + x - 1$. In this case, we can show that 1, *i* and –*i* are the *exact* zeros of *f*,

We invite you to use a surface grapher such as *Master Grapher* [Waits and Demana, 1989], to determine a geometric representation of $f(x) = 4x^4 + 17x^2 + 14x + 64$ and then use the graph to estimate the zeros of *f*. It turns out these are no real zeros. It turns out then $1 \pm 2i$ and $-1 \pm 1.5i$ are good estimations for the four zeros of *f* [Demana and Waits, 1990].

In general, considerable experimentation is needed to estimate the coordinates of the points where a modulo surface touches the *xy*-plane without prior knowledge of the zeros. The algebraic manipulations required are tedious and are a good reason to use a CAS system. A graphical method to determine the real *and* nonreal complex zeros of a function is powerful because there are no formulas for the exact zeros of polynomials of degree 5 or higher.

References

[1] Demana Franklin and Bert K. Waits. Precalculs Mathematics, A Graphic Approach. Reading, MA: Addison-Wesley Publishin Co., 1990

[2] Waits, Bert K. and Franklin Demana, *Master Grapher*, Computer Software for IBM, Apple II and MacIntosh. Reading, MA: Addison-Wesley Publishing Co., 1989

2004 Appendix:

As Bert and Frank noted before, a CAS could be very helpful. Let´s see, what DERIVE and Voyage 200 can do for us in 2004.

The slow CAS - approach:

#10: $u^3 - u^2 + u - 1$

#11: $\text{SUBST}(u^3 - u^2 + u - 1,\ u,\ x + y \cdot i)$

#12: $x^3 - x^2 + x \cdot (1 - 3 \cdot y^2) + y^2 - 1 + i \cdot y \cdot (3 \cdot x^2 - 2 \cdot x - y^2 + 1)$

#13: $\left| x^3 - x^2 + x \cdot (1 - 3 \cdot y^2) + y^2 - 1 + i \cdot y \cdot (3 \cdot x^2 - 2 \cdot x - y^2 + 1) \right|$

#14: $\sqrt{(x^2 + y^2 + 2 \cdot y + 1) \cdot (x^2 + y^2 - 2 \cdot y + 1) \cdot (x^2 - 2 \cdot x + y^2 + 1)}$

Using the mod_surf-function:

#15: $\text{mod\_surf}(x^3 - x^2 + x - 1)$

#16: $\sqrt{(x^2 + y^2 + 2 \cdot y + 1) \cdot (x^2 + y^2 - 2 \cdot y + 1) \cdot (x^2 - 2 \cdot x + y^2 + 1)}$

Finally We compare the CAS-result with the result given in the paper

#17: $(x^3 - 3\cdot x\cdot y^2 - x^2 + y^2 + x - 1)^2 + (3\cdot x^2\cdot y - y^3 - 2\cdot x\cdot y + y)^2$

#18: $(x^2 + y^2 + 2\cdot y + 1)\cdot(x^2 + y^2 - 2\cdot y + 1)\cdot(x^2 - 2\cdot x + y^2 + 1) - ((x^3 - 3\cdot x\cdot y^2 - x^2 + y^2 + x - 1)^2 + (3\cdot x^2\cdot y - y^3 - 2\cdot x\cdot y + y)^2) = 0$

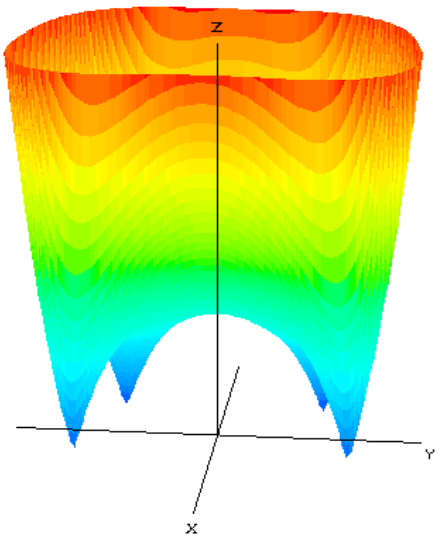
We accept Bert and Frank´s invitation and we produce the "four rooted tooth" together with its contour lines.

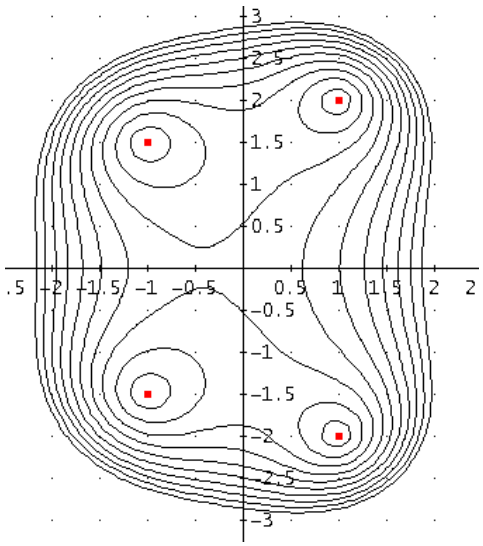
$$z := \sqrt{(16\cdot x^8 + 8\cdot x^6\cdot(8\cdot y^2 + 17) + 112\cdot x^5 + x^4\cdot(96\cdot y^4 + 136\cdot y^2 + 801) + 28\cdot x^3\cdot(17 - 8\cdot y^2) + 2\cdot x^2\cdot(32\cdot y^6 - 68\cdot y^4 - 1247\cdot y^2 + 1186)}$$

$$- 28\cdot x\cdot(12\cdot y^4 - 17\cdot y^2 - 64) + 16\cdot y^8 - 136\cdot y^6 + 801\cdot y^4 - 1980\cdot y^2 + 4096)$$

VECTOR(z = k, k, 0, 200, 20)

$$\begin{bmatrix} 1 & 2 \\ 1 & -2 \\ -1 & 1.5 \\ -1 & -1.5 \end{bmatrix}$$





These are the exact zeros:

SOLUTIONS($4\cdot x^4 + 17\cdot x^2 + 14\cdot x + 64 = 0$, x) = [0.9927139619 + 1.997974745·i, 0.9927139619 − 1.997974745·i, −0.9927139619 − 1.493003327·i, −0.9927139619 + 1.493003327·i]

NSOLUTIONS($4\cdot x^4 + 17\cdot x^2 + 14\cdot x + 64 = 0$, x) = [−0.9927139619 + 1.493003327·i, −0.9927139619 − 1.493003327·i, 0.9927139619 + 1.997974745·i, 0.9927139619 − 1.997974745·i]

Derive offers not only the CAS-manipulations and nice graphic representations, but now Derive 6 offers two more features.

- We can transfer our manipulations to the handheld device and then demonstrate the results.
- We can introduce slider bars to generalize the modulo surfaces and have interesting investigations.

Let´s start with the transfer. We start with the Derive session considering some TI-specials and export the file to the TI – named `modulus` – which results in the text file on the TI.

The Derive file looks like as folllows:

Modulo Surfaces

#1: $f(x) := x^3 - 4 \cdot x$

#2: $|f(x)|$

#3: $mod\_surf(f\_) := |SUBST(f\_, u, x + y \cdot i)|$

#4: $mod\_surf(u^2 + 1)$

#5: $\sqrt{(x^4 + 2 \cdot x^2 \cdot (y^2 + 1) + y^4 - 2 \cdot y^2 + 1)}$

#6: $z1(x, y) := \sqrt{(x^4 + 2 \cdot x^2 \cdot (y^2 + 1) + y^4 - 2 \cdot y^2 + 1)}$

The slow CAS-approach:

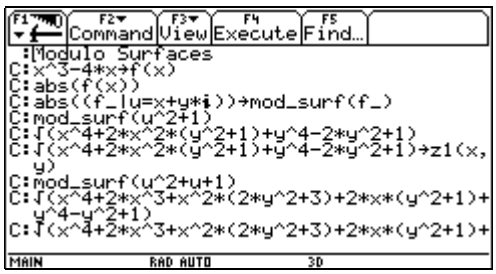
#7: $u^3 - u^2 + u - 1$

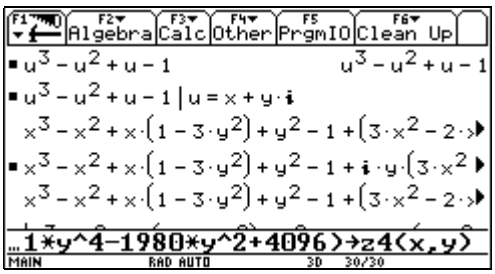
#8: $SUBST(u^3 - u^2 + u - 1, u, x + y \cdot i)$

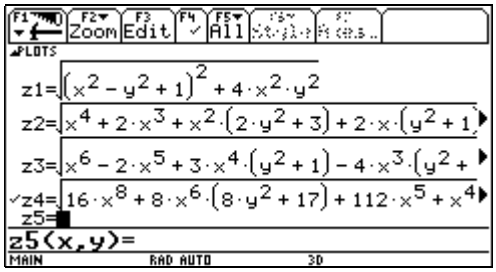
#9: $x^3 - x^2 + x \cdot (1 - 3 \cdot y^2) + y^2 - 1 + i \cdot y \cdot (3 \cdot x^2 - 2 \cdot x - y^2 + 1)$

#10: $\left| x^3 - x^2 + x \cdot (1 - 3 \cdot y^2) + y^2 - 1 + i \cdot y \cdot (3 \cdot x^2 - 2 \cdot x - y^2 + 1) \right|$

After a successful transfer to the Voyage 200 (or TI-92+) we can open the text file `modulus` and run the commands in the Home screen by pressing continuously F4. All steps are performed perfectly and all modulo surfaces are stored in the Y= Editor (which is a z-Editor in 3D-Mode).

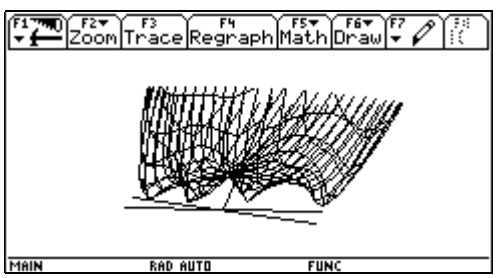


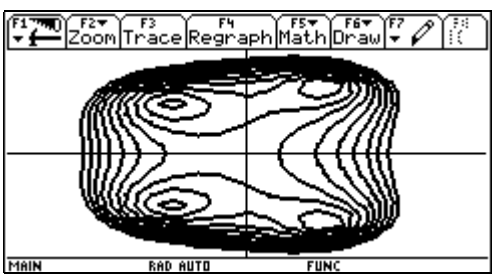




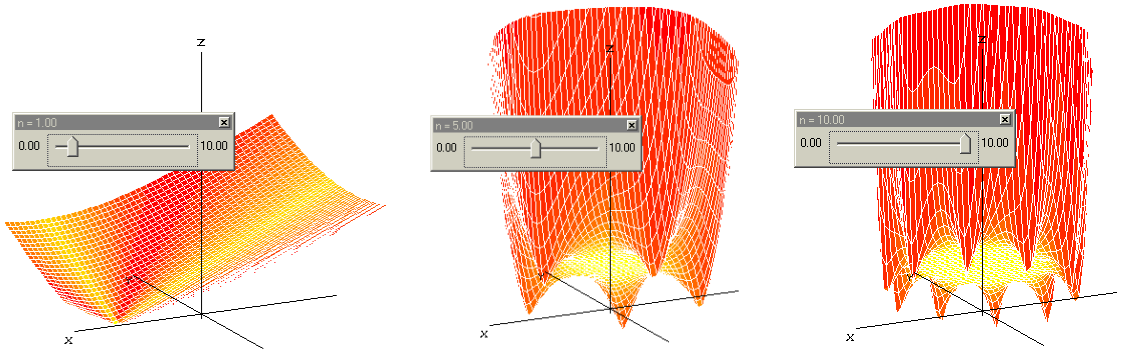
Finally we can plot the surface and the contour lines – but it takes remarkably more time than on the PC.

On the next page you can find exciting investigations using slider bars with Derive 6.

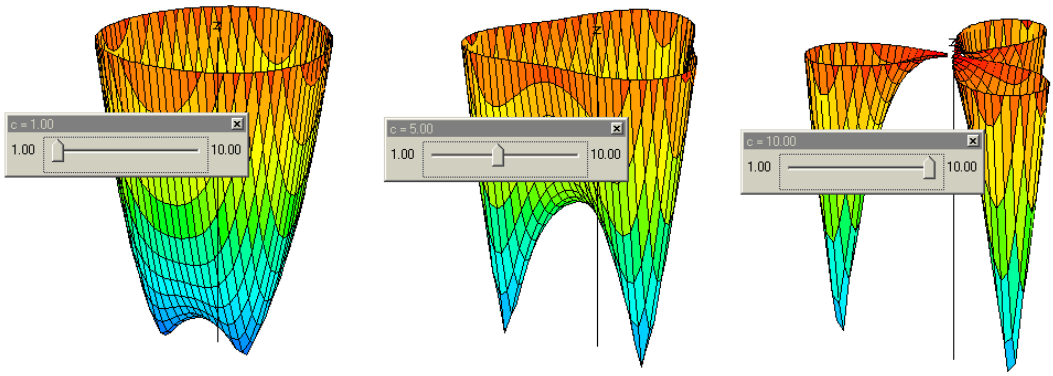




$\text{mod\_surf}(x^n - 1)$	Slider bar for n (n = 1, n = 5 and n = 10) Try $1 \le n \le 2$ with 10 intervals! It´s interesting to follow $2 \le n \le 3$ with 10 intervalls.



$\text{mod\_surf}(x^3 - c)$	Slider bar for c (c = 1, c = 5 and c = 10)

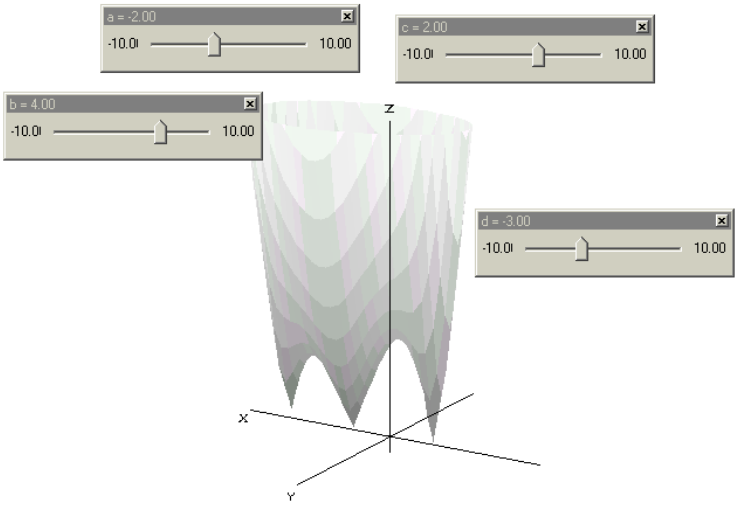


Take a generalized cubic with variable coefficients and investigate its zeros in a 3D-representation. In the case depicted below we have three real roots – all the peaks are on the *x*-axis.

$$\text{mod\_surf}(a \cdot x^3 + b \cdot x^2 + c \cdot x + d)$$

$$\text{SOLVE}(-2 \cdot x^3 + 4 \cdot x^2 + 2 \cdot x - 3 = 0,\ x)$$

$$x = 0.7703185293 \vee x = 2.139726157 \vee x = -0.9100446871$$



Stückweise lineare Funktionen in der FUZZY Logik
Piecewise linear functions in FUZZY Logic

Gerhard Hagen, Austria

So-called FUZZY LOGIC provide clear and usefule examples for piecewise linear functions. By means of FUZZY LOGIC many – especially technical – problems can be modelled in a better way than by classical methods. Particularly in case if qualitative properties must be quantified we prefer soft transitions instead of jumps.

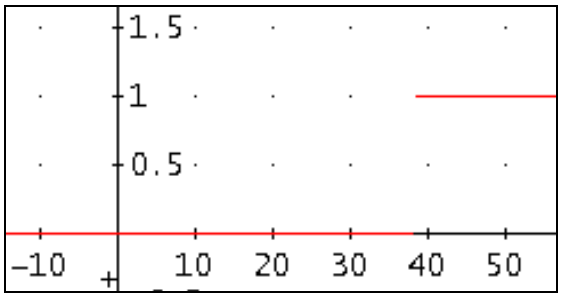
Take the following example from medicine:

If we have to decide if a person has fever using the properties "temperature of the body [C°]" the classical method differs only between "has fever" (starting with how many degrees?) and "has no fever".

We can express this in the following form as a function:

$$z(t) = \begin{cases} 1 & \text{for} \quad t \geq 38.5° \\ 0 & \text{for} \quad t < 38.5° \end{cases}$$

```
z(t) :=
  If t < 38.5
     0
     1
```



In FUZZY LOGIC we define a FUZZY-variable `body-temperature` by expressions *raised temperature* (*erhöhte Temperatur*) RT, *fever* (*Fieber*) FE and "*high temperature*" (*hohes Fieber*) HF. We create a membership-function for each expression as follows:

RT(t) = 0	for $t < 36$	no fever at all
RT(t) = 1	for $37 \leq t < 37.5$	region for "raised"
RT(t) = 0	for $t \geq 38$	"real" fever or even "high temperature"

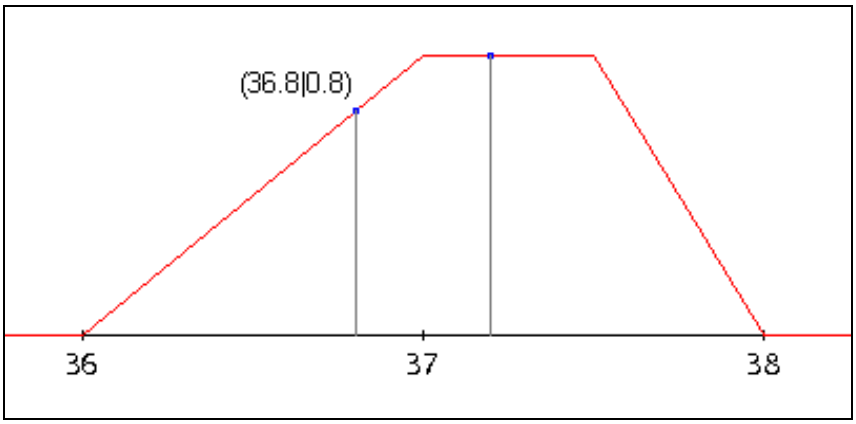
Between 36°C and 37°C there is a transition, which can be described in the easiest way by a linear function.

RT(t) = $t - 36$ for $36 \leq t < 37$

In the same way between 37.5° and 38°

RT(t) = $-2t + 76$ for $37.5 \leq t < 38$

This results in a typical FUZZY – Membership Graph.



Now we have to interpret this graph. We can read off that a temperature of 36.8° can be graded as "*raised temp*" with 80%, its level of membership is 0.8. 37.2°C is "*raised temperature*" at a level of 100%.

For both other expressions *fever* and *high temperature* analoguous definitions must be generated. Hence each measured temperature will have three levels of membership (one for each expression). By using this triple of statements better decisions are possible instead of relying on the 0-1 (YES/NO) assignment in many cases.

Generally spoken a typical (simple) FUZZY-graph has the form of a trapezium. The important positions are named as a, b, c and d. We get the conditions for a generalized "membership function" $z(x)$:

$$z(x)=\begin{cases} 0 & \text{for} \quad x<a \\ \dfrac{x-a}{b-a} & \text{for} \quad a\le x<b \\ 1 & \text{for} \quad b\le x<c \\ \dfrac{x-d}{c-d} & \text{for} \quad c\le x<d \\ 0 & \text{for} \quad x\ge d \end{cases}$$

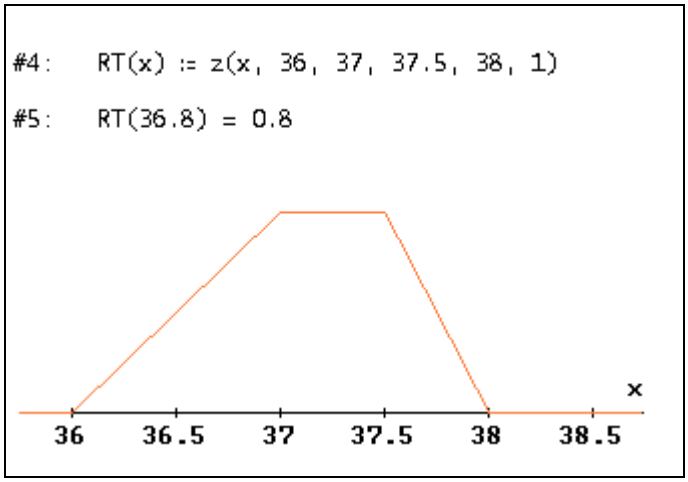
Maximum level of membership h – in example above $h = 1$ – can vary in the interval [0,1] depending on the problem. In this case we have $z(x) = h$ for $b \le x < c$.

This generalization can modelled by nested IFs in Derive (be nested WHENs on the CAS-TI):

```
z(x, a, b, c, d, h) :=
  If x < a
     0
     If x < b
        h·(x - a)/(b - a)
        If x < c
           h
           If x < d
              h·(x - d)/(c - d)
              0
```

$$\text{IF}\left(x<a,\ 0,\ \text{IF}\left(x<b,\ \frac{h\cdot(x-a)}{b-a},\ \text{IF}\left(x<c,\ h,\ \text{IF}\left(x<d,\ \frac{h\cdot(x-d)}{c-d},\ 0\right)\right)\right)\right)$$

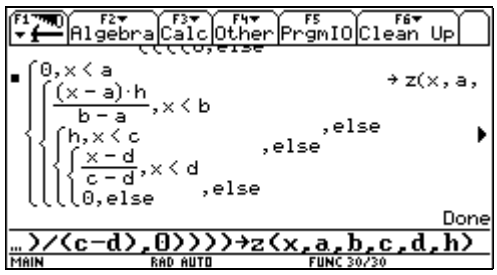
Using this function we redefine RT(x) and we receive the same graph as before:

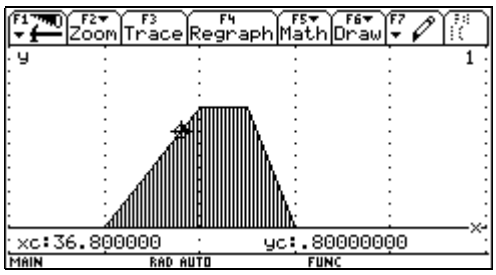


Technical applications are control mechanisms for machines. Substitute the `body temperature` and its expressions by "`distance to the goal`" and "*far away*", "*close*", "*very close*", "*target point*" then you have started modelling the control system for a crane trolley. It is obvious that the modelling process becomes very soon more complicated.

A fine introduction is

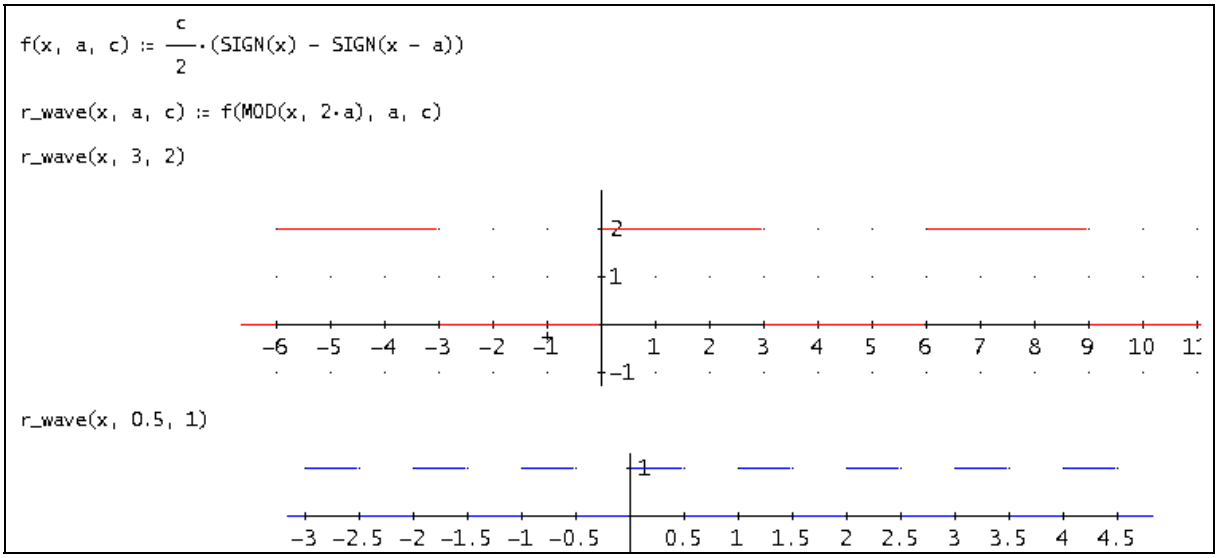
Gerorg HEINRICHS, *Regeln mit Fuzzy*, Naturwissenschaftliche Reihe, 1. Auflage Klett, 1997





****

A student´s question: "How to produce a kind of square wave with Derive without using the built-in `SQUARE_WAVE(x)`-function?"



It is not difficult when using Milton Lesmes´- "Make Periodic Function" from .....

Financial Mathematics II

TVM and Amortization
For Derive 5 & 6

MacDonald R. Phillips
phillipsm@gao.gov
donphillips@starpower.net
February 2004

The two main programs in Financial Mathematics II are
`TVM(n,im,pv,pmt,fv,ppy,cpy,beg,fp)` and
`Amortization(n,im,pv,pmt,bal,ppy,cpy,beg,fp,g,f)`.

Other programs called by the main programs include
`RND(x,n)`, `ID(i,nth,mth)`, `SPPV(i,n)`, `USPV(i,n,beg)`, `SPFV(i,n)`, and `USFV(i,n,beg)`.

The variables are:

- n = number of payments
- im = annual interest or discount rate compounded `cpy` times per year
- pv = present value
- pmt = annuity payment
- fv = future value
- ppy = number of payments per year (cannot be infinity)
- cpy = number of times the interest or discount rate is compounded per year; may be infinity (enter `cpy` as a negative value if using a discount rate)
- beg = if 0 (the default), payments are made at the end of the period; if 1, payments are made at the beginning of the period
- fp = placement of final payment on an annuity due (lease). Default is 0. (If `beg` = 0, `fp` is automatically set to 0. If `beg` = 1, and `fp` = 0, the future value or ballon payment on an annuity-due is at the end of the annuity-due period; if `fp` = 1, the future value or ballon payment is made with the final payment.)
- bal = in the amortization program, bal is an amount, if any, that is added to or substracted from the final payment to determine the final odd payment (default is 0)
- g = number of paymets to group together when computing an amortization schedule (default is 1)
- f = number of payments in the first group (if 0, `f` is set to `g`)

The routines all use the cash flow sign convention. Cash inflows are entered as positive numbers and cash outflows as negative numbers.[1]

The `RND` function rounds the value(s) of a number, vector, or matrix to `n` decimal points; the non-number elements are left alone.

The `ID` function converts an interest/discount rate compounded for `nth` periods per year to an interest/discount rate compouned `mth` periods per year. If `nth` or `mth` are negative, it represents a discount rate. `nth` and `mth` may take on any values, including infinity [1].

The `SPPV` function computes the present value of 1 received `n` periods from now at interest rate `i` per period.

[1] This does not work with the TI-TVM-Solver.

The USPV function computes the present value of an annuity of 1 received for n periods at an interest rate of i per period. The beg variable determines whether the annuity is an annuity-immediate (beg = 0) or an annuity-due (beg = 1).

The SPFV function computes the future value of 1 invested now and received in n periods at an interest rate of i.

The USFV funciton computes the future value of an annuity of 1 invested for n periods at an interest rate of i. The beg variable determines whether the annuity is an annuity-immediate (beg = 0) or an annuity-due (beg = 1).

1. Interest Rate Conversions

The mathematics of interest/discount rate conversion is gone into in detail in *The Theory of Interest*, 2nd ed., by Stephen G. Kellison. The ID(i,nth,mth) function will compute them all. i is the interest rate in decimal form. nth is the current number of compounding periods per year. And mth is the number of compounding periods per year to convert i to. For instance, a 10% interest rate compounded annually is equivalent to a 9.7617... interest rate compounded semi-annually, a 9.5689... interest rate compounded monthly, a 9.531... interest/discount rate compounded continuously, a 9.4932... discount rate compounded monthly, or a 9.0909... discount rate compounded annually, as seen in the table below.

(It is interesting to note that the continuously compounded rate for equivalent interest and discount rates is the same. Continuously compounded rates are also known as the force of interest or the force of discount.)

The interest rate for the financial functions is entered as a percent, not a decimal. And, a discount rate is denoted by a minus sign in front of nth or mth. For instance, a 10% interest rate compounded annually is equivalent to the following interest and discount rates compounded cpy times per year.

```
APPEND([[CpY, Rate]], TABLE(ID(10, 1, n), n, [1, 2, 3, 4, 6, 12, 365, ∞, -365,
      -12, -6, -4, -3, -2, -1]))
```

CpY	Rate
1	10
2	9.761769634
3	9.684034636
4	9.645475633
6	9.607120664
12	9.568968514
365	9.532262476
∞	9.53101798
-365	9.529773701
-12	9.493267863
-6	9.455716974
-4	9.418364129
-3	9.381208154
-2	9.30748215
-1	9.09090909

To go, for instance, from an annually compounded discount rate to an equivalent annually compounded interest rate, simply do this:

ID(9.09090909, −1, 1) = 10

A continuously compounded rate of 11% is equivalent to what nominal rate of interest compounded monthly?

ID(11, ∞, 12) = 11.05057107

The ID function was developed to be used in the financial functions where the number of payments per year is different from the number of interest compounding periods per year.

2. The Time-Value-of-Money: Annuities-Immediate

(Don and I had a "competition" producing a TVM-Solver for Derive. We both were inspired by the Finance-Tool for the TI-handheld. It might be interesting for programmers to compare our both products. Don´s output shows a nice feature with the leading asterisk for the result. My advantage could be that I am introducing global variables, so the results for n, I, pv, pmt and fv are stored under these names and can be used in a subsequent step, which is often necessary for more complex problems. I put Don´s output, my output (blue) and the TI´s output in one row. Unfortunately the Finance-tool on the TI fails in several cases although it is implemented on a CAS-machine!!!. I overcame this problem and programmed my own TVMS (Time-Value-Money-Symbolic Tool). See DNL#49. Josef)

The TVM function will handle almost all general annuities. The only class of annuities it will not compute, that I am aware of, is annuities with continuously paid or received payments; i.e., ppy cannot be set to infinity. The function will compute perpetuities, however. That is, n can be set to infinity.

The best way to see how the function works is to compute some examples. For the unknown to be solved for, simply enter the name of the variable instead of a number. (In Josef´s TVM-tool enter x.)

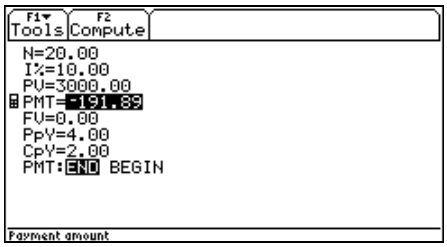
Example 1: A loan of $3,000 is to be repaid with quarterly installments at the end of each quarter for five years. If the rate of interest charged on the loan is 10% convertible semiannually, find the amount of each payment.

TVM(20, 10, 3000, pmt, 0, 4, 2)

N	20
IR(%)	10
PV	3000
*PMT	−191.8875238
FV	0
PpY:	4
CpY:	2
Payment:	END

tvm(20, 10, 3000, x, 0, 4, 2)

N =	20
I% =	10
PV =	3000
PMT =	−191.8875238
FV =	0
PpY =	4
CpY =	2
pmt :	END
interest:	INTEREST



The payment is -191.89. (The variable solved for is preceeded by an asterisk (*).)

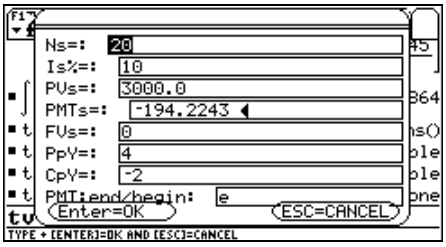
If you were using a discount rate of 10% convertible semiannully, instead, the payment would be:

TVM(20, 10, 3000, pmt, 0, 4, -2)

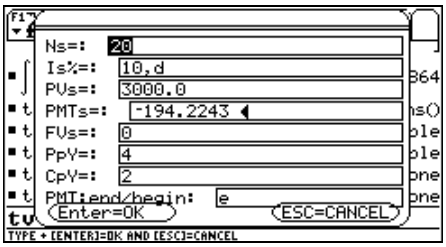
N	20
DR(%)	10
PV	3000
*PMT	-194.2243477
FV	0
PpY:	4
CpY:	2
Payment:	END

tvm(20, 10, 3000, x, 0, 4, 2, e, d)

N =	20
I% =	10
PV =	3000
PMT =	-194.2243477
FV =	0
PpY =	4
CpY =	2
pmt :	END
interest:	DISCOUNT



TVMS-Solver →



Notice that IR(%) has been replaced by DR(%) indicating that a discount rate was used instead of an interest rate. The reason for the difference in payments is that a discount rate of 10% compounded semiannually is equivalent to an interest rate greater than 10%. It is, in fact, equal to an interest rate of 10.526...% compounded semiannually.

ID(10, −2, 2) = 10.52631578

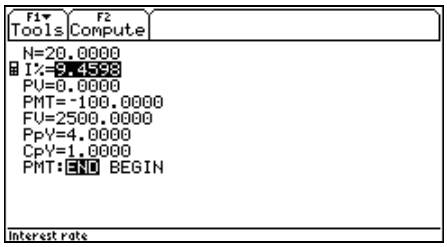
Example 2: At what annual effective rate of interest will payments of $100 at the end of every quarter accumulate to $2,500 at the end of five years?

TVM(5·4, im, 0, −100, 2500, 4, 1)

N	20
*IR(%)	9.459777745
PV	0
PMT	−100
FV	2500
PpY	4
CpY	1
Payment	END

tvm(20, x, 0, −100, 2500, 4)

N =	20
I% =	9.459777745
PV =	0
PMT =	−100
FV =	2500
PpY =	4
CpY =	1
pmt :	END
interest:	INTEREST



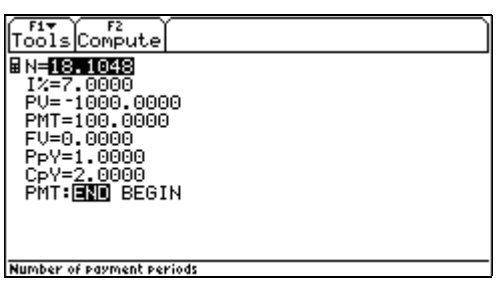
Example 3: An investment of $1,000 is used to make payments of $100 at the end of each year for as long as possible with a smaller final payment to be made at the time of the last regular payment. If interest 7% convertible semiannually, find the number of payments and the amount of the total final payment.

TVM(n, 7, −1000, 100, 0, 1, 2)

*N	18.10481933
IR(%)	7
PV	−1000
PMT	100
FV	0
PpY	1
CpY	2
Payment	END

tvm(x, 7, −1000, 100, 0, 1, 2)

N =	18.10481933
I% =	7
PV =	−1000
PMT =	100
FV =	0
PpY =	1
CpY =	2
pmt :	END
interest:	INTEREST



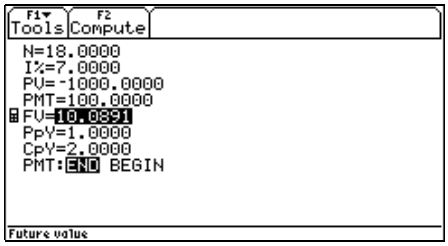
Thus, 18 payments are made. The amount to be added to the final payment is:

TVM(18, 7, -1000, 100, fv, 1, 2)

N	18
IR(%)	7
PV	-1000
PMT	100
⋆FV	10.08905079
PpY	1
CpY	2
Payment	END

tvm(18, 7, -1000, 100, x, 1, 2)

N =	18
I% =	7
PV =	-1000
PMT =	100
FV =	10.08905079
PpY =	1
CpY =	2
pmt :	END
interest:	INTEREST



The total final payment is therefore \$100 + \$10.09 = \$110.09.

Example 4: At what annual effective rate of interest is the present value of a series of payments of \$1 every six months forever, with the first payment made immediately, equal to \$10?

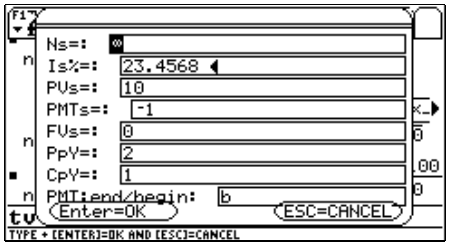
TVM(∞, im, 10, -1, 0, 2, 1, 1)

N	∞
⋆IR(%)	23.45679012
PV	10
PMT	-1
FV	0
PpY	2
CpY	1
Payment	BEGIN

tvm(∞, x, 10, -1, 0, 2, 1, b)

N =	∞
I% =	23.45679012
PV =	10
PMT =	-1
FV =	0
PpY =	2
CpY =	1
pmt :	BEGIN
interest:	INTEREST

∞ requires TVMS!



The FINANCE-Tool which is implemented on the CAS-TI devices is a great tool, but it fails for some – important – cases. Originally it was made for the graphic calculators – without any use of CAS – and unfortunately it was transferred to the CAS-calculators without considering that there are more possibilities than on the TI-83/84 family. So it is not possible to have an infite number of payments – see the example given above – and it is also not possible to work with payment periods longer than a year. 1/2 or .5 is not accepted for PpY which is necessary to solve Example 5. I wrote a TVMS (TIME-VALUE-MONEY-Symbolical-Solver), which overcomes all those problems. So some of the TI-screen shots don´t show the original TVM-Solver, but my TVMS. Josef

Example 5: Find the accumulated value 18 years after the first payment is made of an annuity of which there are 8 payments of \$2,000 each made at two-year intervals. The nominal rate of interest convertible semiannually is 7%.

If we take the 8 payments at the beginning of each two-year period, the future value at the end of 16 years is:

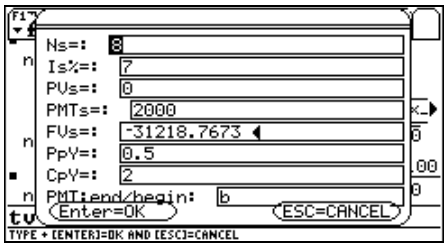
(Note: Since the payments are every 2 years, ppy is set to 0.5.)

TVM(8, 7, 0, 2000, fv, 0.5, 2, 1)

N	8
IR(%)	7
PV	0
PMT	2000
∗FV	-31218.76725
PpY	0.5
CpY	2
Payment	BEGIN

$$tvm\left(8,\ 7,\ 0,\ -2000,\ x,\ \frac{1}{2},\ 2,\ b\right)$$

N =	8
I% =	7
PV =	0
PMT =	-2000
FV =	31218.76725
PpY =	0.5
CpY =	2
pmt :	BEGIN
interest:	INTEREST



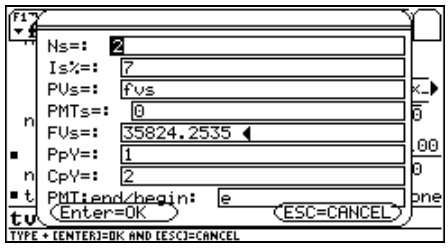
Takeing the computed FV forward two more years gives:

TVM(2, 7, -31218.76725, 0, fv, 1, 2)

N	2
IR(%)	7
PV	-31218.76725
PMT	0
∗FV	35824.25347
PpY	1
CpY	2
Payment	END

tvm(2, 7, -fv, 0, x, 1, 2)

N =	2
I% =	7
PV =	-31218.76725
PMT =	0
FV =	35824.25347
PpY =	1
CpY =	2
pmt :	END
interest:	INTEREST



The answer is $35,824.25.

3. Amortization of Annuities-Immediate

The amortization method is one of two general methods of repaying a loan; the other is the sinking fund method. In the amortization method the borrower repays the lender by means of installment payments at periodic intervals. In the sinking fund method the borrower repays the lender by means of one lump-sum payment at the term of the loan. The borrower pays interest on the loan in installments over this period. It is assumed that the borrower make periodic payments into a fund, called a sinking fund, which will accumulate to the amount of the loan to be repaid at the end of the term of the loan. The `Amortization()` routine here cannot be used for sinking funds.

Example: You have just purchased a $25,000 car to be repaid over 36 months at an interest rate of 7.5% compounded monthly. Compute the monthly payment, the additional payment to be made at the time of the final payment, and the amortization schedule.
The amortization program rounds all values to 2 decimal places, i.e., dollars and cents.

```
TVM(36, 7.5, 25000, pmt, 0, 12, 12)
```

N	36
IR(%)	7.5
PV	25000
*PMT	-777.655454
FV	0
PpY	12
CpY	12
Payment	END

```
tvm(36, 7.5, 25000, x, 0, 12, 12)
```

N =	36
I% =	7.5
PV =	25000
PMT =	-777.6554540
FV =	0
PpY =	12
CpY =	12
pmt :	END
interest:	INTEREST

The regular payment is $777.66. The additional payment to be made with the final payment is:

```
TVM(36, 7.5, 25000, -777.66, fv, 12, 12)
```

N	36
IR(%)	7.5
PV	25000
PMT	-777.66
*FV	0.1828912008
PpY	12
CpY	12
Payment	END

```
tvm(36, 7.5, 25000, rd(pmt, 2), x, 12, 12)
```

N =	36
I% =	7.5
PV =	25000
PMT =	-777.66
FV =	0.1828912008
PpY =	12
CpY =	12
pmt :	END
interest:	INTEREST

The addition payment is $0.18 for a final total payment of -$777.48.

The amortization schedule is:

```
Amortization(36, 7.5, 25000, -777.66, 0.18, 12, 12)
```

Time	Payment	Interest	Principal	Balance
0	0	0	0	25000
1	-777.66	-156.25	-621.41	24378.59
2	-777.66	-152.37	-625.29	23753.3
3	-777.66	-148.46	-629.2	23124.1
4	-777.66	-144.53	-633.13	22490.97
.....				
.....				
33	-777.66	-19.14	-758.52	2303.94
34	-777.66	-14.4	-763.26	1540.68
35	-777.66	-9.63	-768.03	772.65
36	-777.48	-4.83	-772.65	0
Total:	-27995.58	-2995.58	-25000	

Total payments are $27,995.58 of which $2,995.58 is interest. Note the final payment of -$777.48. But supposed you wanted to know the payments and interest on a yearly basis for tax purposes.

Amortization(36, 7.5, 25000, -777.66, 0.18, 12, 12, 0, 0, 12)

Payments	∑Payments	∑Interest	∑Principal	Balance
1 - 12	-9331.92	-1613.27	-7718.65	17281.35
13 - 24	-9331.92	-1014.02	-8317.9	8963.45
25 - 36	-9331.74	-368.29	-8963.45	0
Total:	-27995.58	-2995.58	-25000	

Oops, you actually made only 5 payments in the first year. What would the yearly schedule look like then?

Amortization(36, 7.5, 25000, -777.66, 0.18, 12, 12, 0, 0, 12, 5)

Payments	∑Payments	∑Interest	∑Principal	Balance
1 - 5	-3888.3	-742.18	-3146.12	21853.88
6 - 17	-9331.92	-1369.01	-7962.91	13890.97
18 - 29	-9331.92	-750.82	-8581.1	5309.87
30 - 36	-5443.44	-133.57	-5309.87	0
Total:	-27995.58	-2995.58	-25000	

4. Time-Value-of-Money and Amortization: Annuities-Due (Leases)

The payments on an annuity-due (e.g., a lease) come at the beginning of a period, not at the end of a period like a regular loan or mortgage (annuity-immediate). For instance, on a 36 month annuity-due the first payment is made immediately and the last payment is made at the beginning of the 36th month, not its end. At times there may be an adjustment to the final payment that is made either with the final payment, or at the end of the annuity's term. Both the `TVM` and `Amortization` programs can handle either situation through the `fp` variable. If `beg` is set to 1 (payments at beginnign of the period) and `fp` is 0 (zero), the adjust-ment or ballon payment is made at the end of the annuity's term; if `fp` is 1, the adjustment is made with the final payment.

Example 1: Find the payment on an annuity-due of 36 months, an interest rate of 10% com-pounded monthly, and a present value of $10,000. What adjustment must be made to the final payment when the payment is rounded to 2 decimal places? To have the adjustment made to the final payment `fp` is set to 1 and remember to set `beg` to 1 also.

TVM(36, 10, 10000, pmt, 0, 12, 12, 1)

N	36
IR(%)	10
PV	10000
*PMT	-320.0051622
FV	0
PpY	12
CpY	12
Payment	BEGIN

TVM(36, 10, 10000, -320.01, fv, 12, 12, 1, 1)

N	36
IR(%)	10
PV	10000
PMT	-320.01
*FV	0.2021298824
PpY	12
CpY	12
Payment	BEGIN

The adjustment is $0.20. The final payment is therefore $-320.01 + 0.20 = $319.81.

Compute the amortization schedule for this annuity-due.

```
Amortization(36, 10, 10000, -320.01, 0.2, 12, 12, 1, 1)
```

Time	Payment	Interest	Principal	Balance
0	-320.01	0	-320.01	9679.99
1	-320.01	-80.67	-239.34	9440.65
2	-320.01	-78.67	-241.34	9199.31
3	-320.01	-76.66	-243.35	8955.96
4	-320.01	-74.63	-245.38	8710.58
32	-320.01	-10.45	-309.56	944.07
33	-320.01	-7.87	-312.14	631.93
34	-320.01	-5.27	-314.74	317.19
35	-319.81	-2.64	-317.17	0.02
Total:	-11520.16	-1520.18	-9999.98	

Notice that the final payment is $0.20 less than the prior payments.

Example 2: Compute the ballon payment to the above annuity-due if the regular payments are only $300 a month, with the ballon payment coming at the end of the payment periods. In this case `fp` set to zero (its default).

```
TVM(36, 10, 10000, -300, fv, 12, 12, 1)
```

N	36
IR(%)	10
PV	10000
PMT	-300
*FV	-842.8175443
PpY	12
CpY	12
Payment	BEGIN

The ballon payment is $842.82 made at the end of the 36th month. Compute the amortization schedule.

```
Amortization(36, 10, 10000, -300, -842.82, 12, 12, 1)
```

Time	Payment	Interest	Principal	Balance
0	-300	0	-300	9700
1	-300	-80.83	-219.17	9480.83
35	-300	-9.39	-290.61	835.84
36	-842.82	-6.97	-835.85	-0.01
Total:	-11642.82	-1642.81	-10000.01	

Notice that there is an extra payment with this annuity-due, a 37th payment which comes at the end of period 36.

Financial Mathematics III will cover net present values, internal rates of return, and modified internal rates of return for uneven cash flows.

If you have any questions, please feel free to send me an email.

I wanted to test Don´s tool with one or the other more examples from my times as school teacher at a College for Business Administration. Josef

This is the first example:

If I had claim to receive 2500 € for three years at the begin of each quarter and I wished to change this in monthly payments which should run forever. How long had I to wait for the first payment at a discount rate of 8.5% compounded quarterly.

In a space saving form it could look as follows:

$$(TVM(12,\ 8.5,\ pv,\ -2500,\ 0,\ 4,\ -4,\ 1))_{3} = [*PV,\ 26730.6286021387]$$

$$(TVM(\infty,\ 8.5,\ pv,\ -500,\ 0,\ 12,\ -4,\ 1))_{3} = [*PV,\ 70085.8487554317]$$

$$(TVM(n,\ 8.5,\ 26730.6286021387,\ 0,\ -70085.8487554317,\ 1,\ -4))_{1} = [*N,\ 11.2192075194681]$$

I had to wait 11.219 years. (And this is the correct result!)

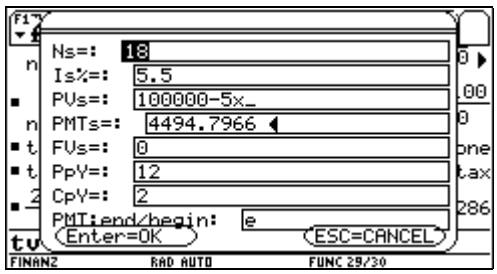
And a second one (DNL#49):

A debt of 100 000 should be paid back by 18 monthly payments due at the end of the months and one payment with an amount of 5 payments which is due immediately. How much is to pay now at an interest rate of 5.5% compounded semiannually?

$$TVM(18,\ 5.5,\ -100000 + 5*pmt,\ pmt,\ 0,\ 12,\ 2)$$

N	18
IR(%)	5.5
*PV	4494.79664326628
PMT	pmt
FV	0
PpY	12
CpY	2
Payment	END

$$RND(5*4494.79664326628,\ 2) = 22473.98$$



The extra payment is 22473.98.

Try to solve this problem with the CAS-TI built in TVM-Solver. I´m quite sure that you will fail!!

Amortization table as an example for an iterative process (Josef):

For teaching financial mathematics it might be useful to introduce an amortization table as a meaningful application of ITERATES. See one possible way to tackle Don´s example from above:

#1: Notation := Decimal

#2: $rd(x) := \frac{ROUND(x,\ 0.01)}{100}$

#3: title := [Period, Balance, Interest, Principal, Payment]

#4: empty := [..., ..., ..., ..., ...]

#5: r0(l_) := [0, l_, - - -, - - -, - - -]

#6: $\left[paym := 777.66,\ i := \frac{0.075}{12}\right]$

#7: $plan1 := ITERATES\left[\begin{matrix} z_1 + 1, & rd(z_2 - (paym - z_2 \cdot i)), & rd(z_2 \cdot i), & rd(paym - z_2 \cdot i), & rd(paym) \end{matrix}\right], z, r0(25000), 36)$

The important step is given in expression #7. It contains the whole "history" of the table.

The last rows of simplified expression #7 are:

$$\begin{bmatrix} 34 & 1540.68 & 14.4 & 763.26 & 777.66 \\ 35 & 772.65 & 9.63 & 768.03 & 777.66 \\ 36 & -0.18 & 4.83 & 772.83 & 777.66 \end{bmatrix}$$

We develop the real last row #36 (derived from row #35) and finally put all together to have a short form of the table containing all important information.

#9: lrow := [36, 0, rd(772.65·i), 772.65, rd(772.65·(1 + i))]

#10: APPEND([title], plan1 ROW [1, 2, 3], [empty], [empty], plan1 ROW [35, 36], [lrow])

Period	Balance	Interest	Principal	Payment
0	25000	- - -	- - -	- - -
1	24378.59	156.25	621.41	777.66
2	23753.3	152.37	625.29	777.66
...	...	...	...	...
...	...	...	...	...
34	1540.68	14.4	763.26	777.66
35	772.65	9.63	768.03	777.66
36	0	4.83	772.65	777.48

According to our textbooks I didn´t use negative numbers for ordinary payments, interests and balances. It would be easy work to change this and to add the sums in an additional row.

Having explained how an amortization table works it is time to invite you to use Don´s excellent tool as a "Black Box" for solving extended problems. Josef

Der Roboter / The Robot

Heinz Rainer Geyer
Gutenbergschule Wiesbaden
März 2004

Mit den 3D-Einheitsvektoren baue ich zuerst einen (Basis-) Quader auf:

Using 3D unit vectors I build a (base-) cuboid:

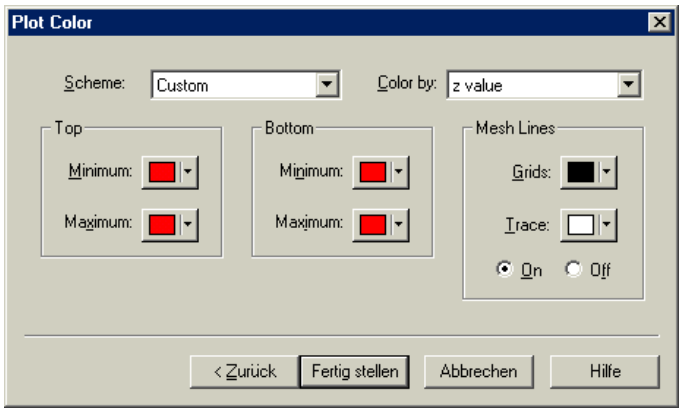
```
[o := [0, 0, 0], x := [1, 0, 0], y := [0, 1, 0], z := [0, 0, 1]]
```

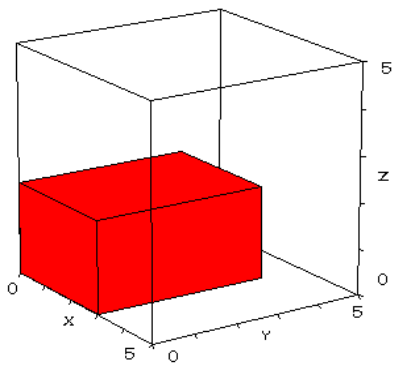
$$\text{cube}(a, b, c) := \begin{bmatrix} o & o + c\cdot z \\ a\cdot x & a\cdot x + c\cdot z \\ a\cdot x + b\cdot y & a\cdot x + b\cdot y + c\cdot z \\ b\cdot y & b\cdot y + c\cdot z \\ o & o + c\cdot z \\ o + c\cdot z & o + b\cdot y + c\cdot z \\ a\cdot x + c\cdot z & a\cdot x + b\cdot y + c\cdot z \\ a\cdot x & a\cdot x + b\cdot y \\ o & o + b\cdot y \end{bmatrix}$$

Teste den Quader mit folgendem Beispiel: Quader mit einer Ecke im Koordinatenursprung und den Seitenlängen 3, 4 und 2

Test the cuboid-function by creating a base cuboid with sides 3, 4 and 2:

```
#3:   cube(3, 4, 2)
```





Simplify expression #3 to find out how to produce a solid with a surface formed by polygons (or rectangles like in this application). Compare the settings for Plot Color to obtain a cuboid showing one single colour.

Folgende Abbildungen werden benötigt / *We need the following mappings*:
- Verschiebung / *Translation* `(trans(obj, vec)`
- Drehungen um die Achsen / *Rotations* `(rotate_i(obj, angle)`

#4: $\mathrm{trans}(obj, vec) := \mathrm{VECTOR}(\mathrm{VECTOR}(obj_{i,j} + vec, j, 1, \mathrm{DIMENSION}(obj_i)), i, 1, \mathrm{DIMENSION}(obj))$

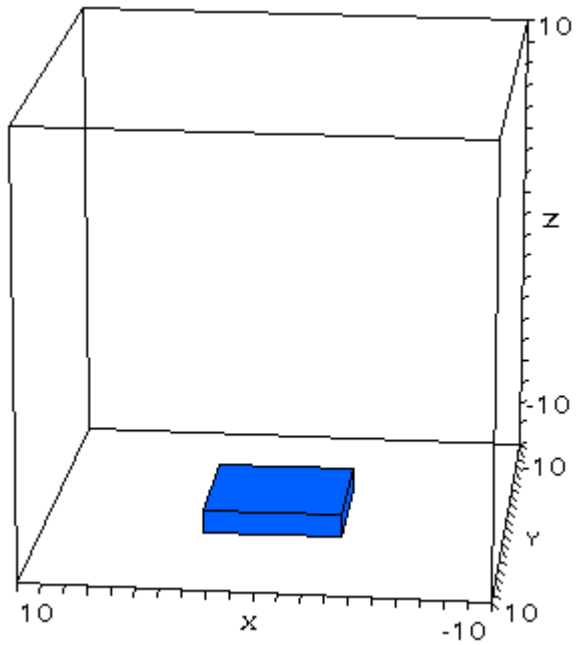
#5: $\mathrm{rotate\_x}(obj, \alpha) := \mathrm{VECTOR}\left(obj_i \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathrm{COS}(\alpha) & \mathrm{SIN}(\alpha) \\ 0 & -\mathrm{SIN}(\alpha) & \mathrm{COS}(\alpha) \end{bmatrix}, i, 1, \mathrm{DIMENSION}(obj)\right)$

#6: $\mathrm{rotate\_y}(obj, \alpha) := \mathrm{VECTOR}\left(obj_i \cdot \begin{bmatrix} \mathrm{COS}(\alpha) & 0 & \mathrm{SIN}(\alpha) \\ 0 & 1 & 0 \\ -\mathrm{SIN}(\alpha) & 0 & \mathrm{COS}(\alpha) \end{bmatrix}, i, 1, \mathrm{DIMENSION}(obj)\right)$

#7: $\mathrm{rotate\_z}(obj, \alpha) := \mathrm{VECTOR}\left(obj_i \cdot \begin{bmatrix} \mathrm{COS}(\alpha) & \mathrm{SIN}(\alpha) & 0 \\ -\mathrm{SIN}(\alpha) & \mathrm{COS}(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}, i, 1, \mathrm{DIMENSION}(obj)\right)$

Der Roboter steht auf einer 6×6×1 Grundplatte:
Our robot is fixed on a 6×6×1 base-plate:

#8: `platte := trans(cube(6, 6, 1), - [3, 3, 10])`

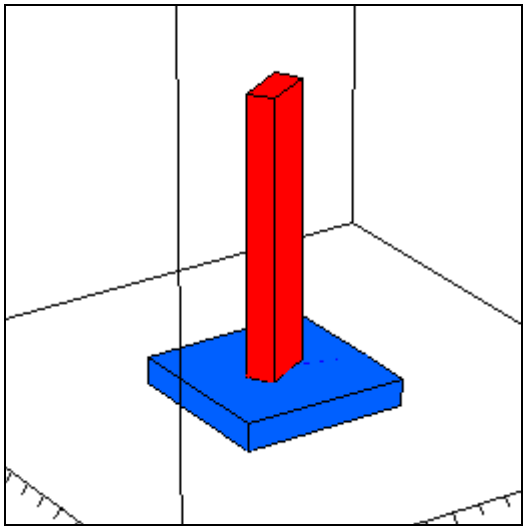


Sein Körper ist um die z-Achse drehbar.
Its body can be rotated about the vertical axis.

#9: `body := trans(cube(1, 2, 10), - [0.5, 1, 9])`

#10: `body_rot(α) := rotate_z(body, α)`

#11: `body_rot(120°)`



Er besitzt einen Arm, der um die x-Achse drehbar ist.
It needs an arm which can be rotated about the x-axis.

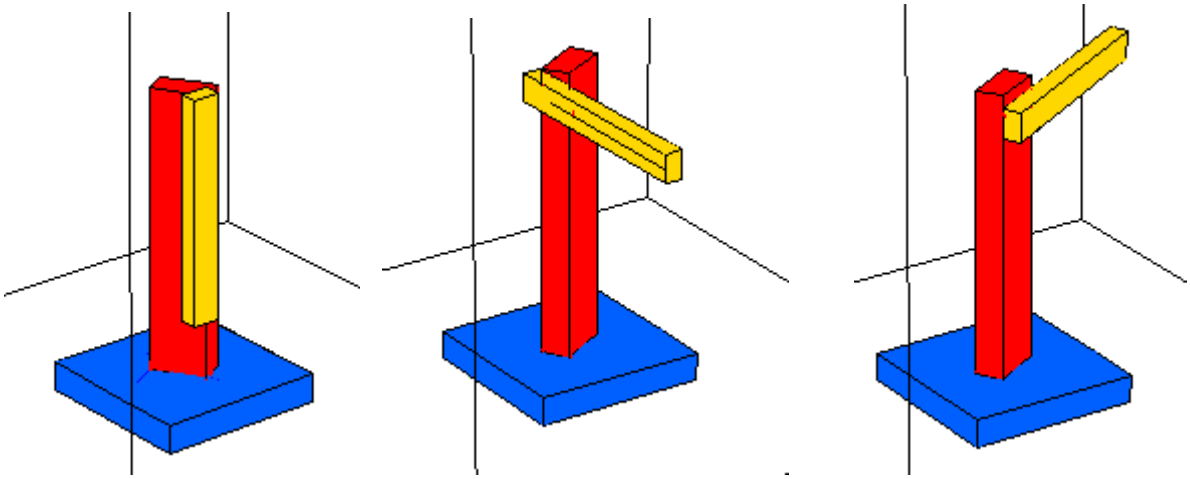
```
#12:  arm := trans(cube(0.6, 1, 8), - [-0.5, 0.5, 7])

#13:  arm_rot(β) := rotate_x(arm, β)

#14:  arm_rot(90°)

#15:  robot_0(α, β) := [platte, body_rot(α), rotate_z(arm_rot(β), α)]

#16:  robot_0(120°, 90°)
```



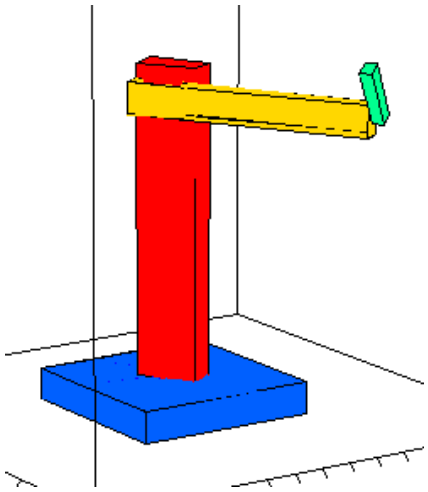
The pictures show the arm in its base position, then rotated by 90° and finally fixed to the body and rotated by 90° (including a 120° rotation of the body).

Ohne Hand kann so ein Roboter nicht greifen. Die Hand kann sich wieder um die z-Achse drehen:
A robot without a hand is useless. The hand can rotate about the z-axis.

Zusammengesetzt ergibt sich der robot_1() mit den Drehwinkeln für Körper, Arm und Hand.
Composed we obtain robot_1() *with rotation angles for* body, arm *and* hand.

```
robot_1(α, β, γ) := [platte, body_rot(α), rotate_z(arm_rot(β), α),
  rotate_z(rotate_x(hand_rot(γ), β), α)]
```

```
robot_1(120°, 90°, 60°)
```

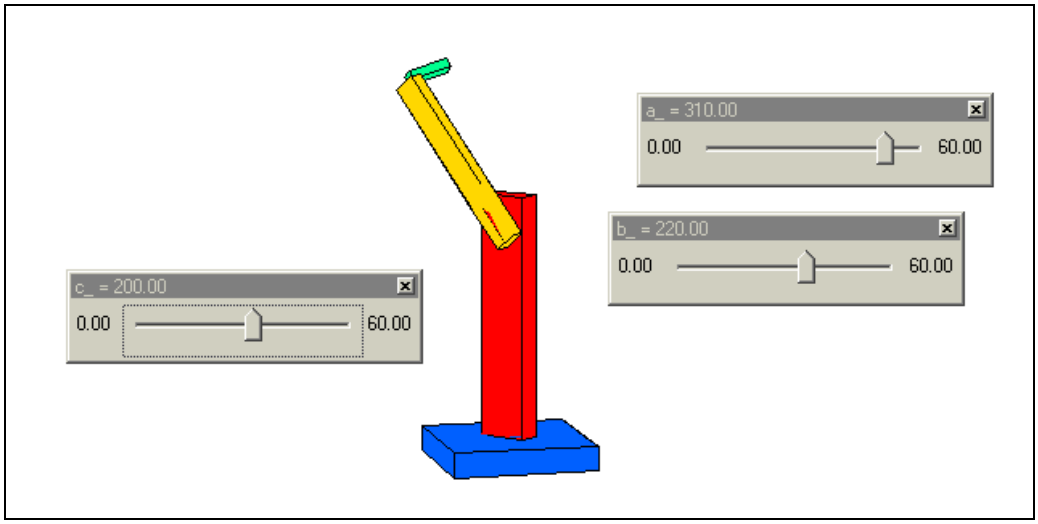


Wir arbeiten im Gradmaß weil Derive 6 für die Schieberegler (noch) nicht π akzeptiert und führen die Winkel a_, b_ und c_ als variable Größen ein und zeichnen den beweglichen Roboter.

We work with degrees, because Derive 6 does not accept (until now!) π as bounds for the slider bars, introduce angles a_, b_ and c_ as variables and plot the animated robot.

```
#26:  robot_2(a_, b_, c_) := [platte, body_rot(a_·1°),
        rotate_z(arm_rot(b_·1°), a_·1°), rotate_z(rotate_x(hand_rot(c_·1°),
        b_·1°), a_·1°)]

#27:  robot_2(a_, b_, c_)
```

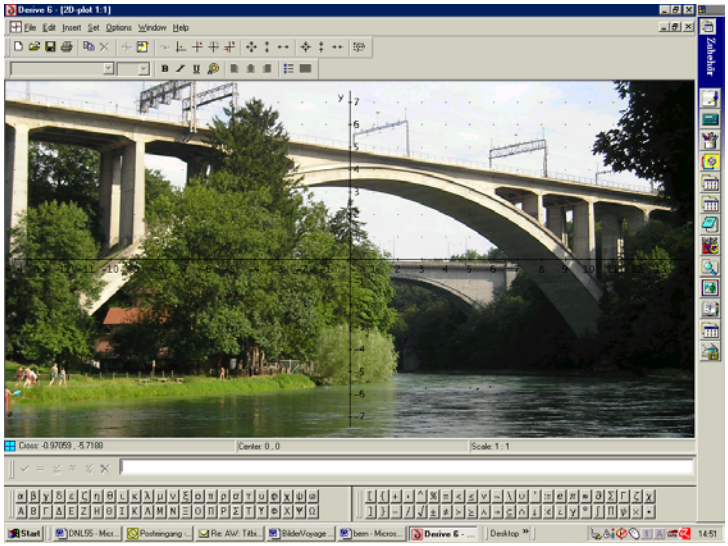


You can be sure that your students will add a second arm, a nose, ET´s finger, ...

Striking Backgrounds – Modelling with Functions

Tania Koller, Vienna

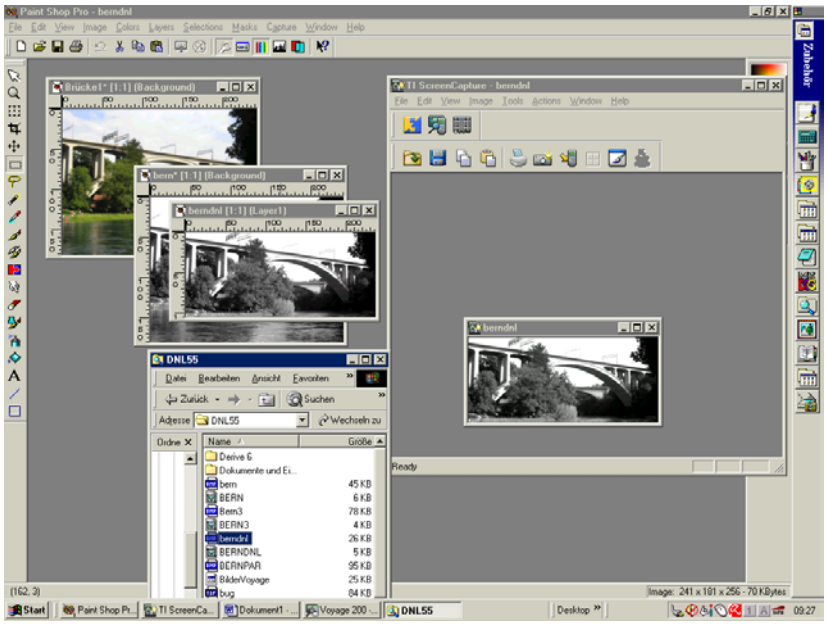
Derive 6 offers the possibility to load pictures in the background of the 2D- and 3D-Plot Windows. I took some pictures in Switzerland with some of the nice bridges in Bern. The students should find out a function to describe the form of the arc.



This is wonderful, but I have several classes working with the Voyage 200 and I wanted to present the same task to them. TI-Connect makes this possible. I found a respective article in the TI-News[1] and then I tried.
This is my recipe:

1. Convert your picture into the *.bmp-format.
2. I work with Microsoft Office Picture Manager (any other Graphics Program will do!)
3. Edit picture
   Color > Saturation –100, Contrast -19
   Resize in a ratio 239 × 104 pixel for Voyage200 (159 × 77 for TI-89)
   I took 1024:446, I resized to 24% of original measures
4. Start TI Connect
5. Open TI Screen Capture → get Screen, click on the file and drag it using the left mouse button into the TI Screen Capture window, then save as *.v2i for Voyage 200 or *.9xi for den TI92+ or *.89i for the TI-89 family.
6. Activate *.v2i in Explorer, right mouse button → Send to TI Device

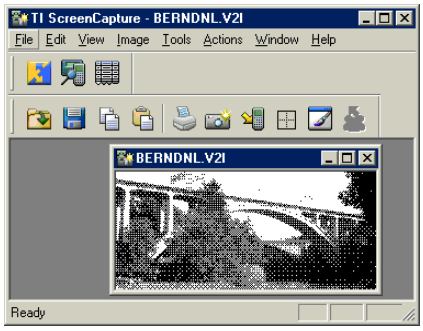
By inserting Tania´s contribution I could easily to follow her clear instructions (using Paint Shop Pro as Graphics Program). I would like to illustrate the process by presentig some screen shots.

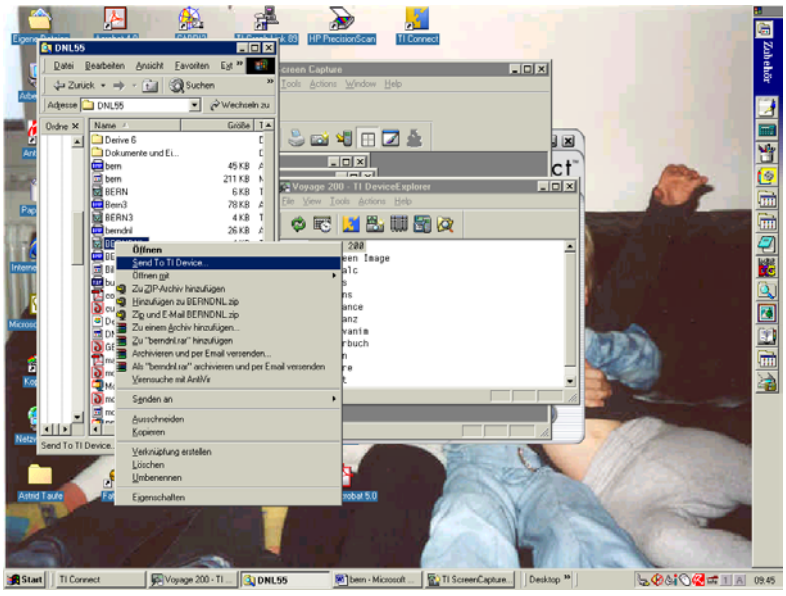


You can see the original picture and then the resized one (I converted into a gray scale graphic). Now we are ready for Screen Capture:

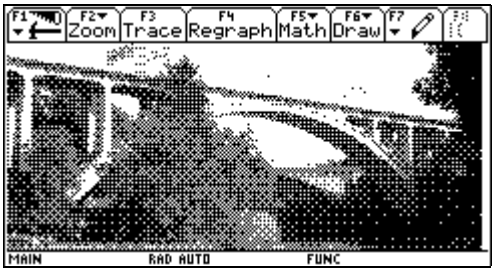
[1] M.Falb, Wie die Bilder auf den Taschencomputer kommen, TI-Nachrichten 1/04

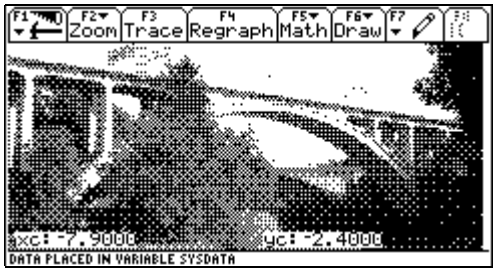
I saved the picture and sent it to the TI. First to the V 200 and then I repeated the process for TI-89.

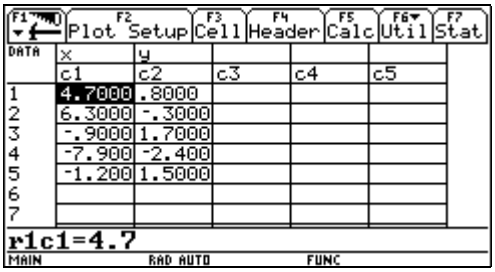


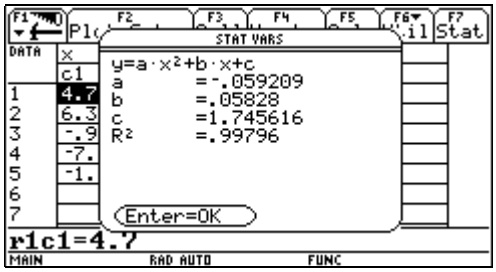


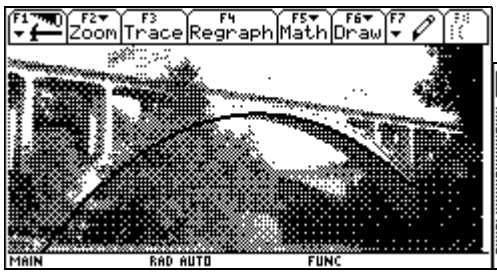
These are the results: The loaded bridge on the V200-screen. I copied some coordinates of the arch into the sysdata - data sheet and performed a quadratic regression – according to Tania´s advice. The last screen shots show is Tania´s pictures of the Aaare River Bridgeon my TI-89 and on my TI-83+

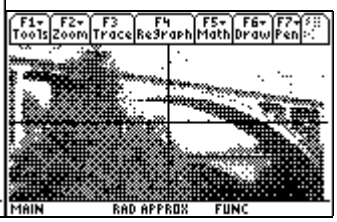












Voyage 200 / TI-92 TI-89 TI-84

The same can be done with Derive 6 and offers a rich variety of problems for our students. I would be happy if you could provide other exciting background pictures which provoke thinking in functions, Tania

Titbits from Algebra and Number Theory (28)

by Johann Wiesenbauer, Vienna

This time I want to go to battle again for what I call "indexfree programming". In my experience most people handle lists in Derive as they would handle arrays in other programming languages, where this concept actually exists. In those languages the time to access an element in an array is independent from its location within the array. This is no longer true in Derive, where you are actually dealing with linked lists. In particular, you should always bear in mind that due to the use of pointers elements at the left end of a list can be far more easily accessed than elements at its right end. If the list is short this doesn't matter, but if the list is long it surely does and you should be aware of this special Derive feature. That this idea is still new to some people I have also seen in some lectures using Derive on the recent Derive-conference in Montreal. (On this occasion, congratulations and many thanks to Michel Beaudin and his people for doing a fabulous job in organizing this conference. Wonderful memories come back while writing these lines!)

Let's take as an example though a program by Rüdeger Baumann in the DNL #52, p 49-50, on the so-called Josephus problem and I hope he doesn't mind. After all, he seemed to be very unhappy himself about the performance of his program and has posed it as a challenge to write a better version of it. Hence, let's do exactly that!

The Josephus problem in its most general starts with n people numbered 1 to n around a circle and it is assumed that every s-th person is killed until nobody is left. Rüdeger was asking for the "Josephus permutation", i.e. for the list of the numbers of the killed people in exactly the order they were killed. For example, if n=10 and s=2 then this list is [2,4,6,8,10,3,7,1,9,5].

Now take a look at his program, which is listed on the next page for the sake of easy reference. What's wrong with it from a programming point of view? Well, you might notice that the only call of append() at the beginning serves no purpose, but does no harm either. You might also notice that there a lot of global variables like v, z, i, j , r, which should be definitely local variables, i.e. they should appear in the list of parameters of the function definition. There are a lot of other points , which deserve to be mentioned here, but the question remains: Why is the program so "terribly slow" quoting Rüdeger himself? The simple answer is: Because of the frequent use of indices, namely r and z in the program – a deadly sin for a Derive programmer aiming at a good performance!

```
Josef(n, s, Liste := []) :=
   Prog
     v := APPEND([1, ..., n])
     z := n
     i := 0
     Loop
       i :+ 1
       If i > n
          RETURN REVERSE(Liste)
       j := 0
       Loop
         j := j + 1
         If j > s exit
         r := IF(z < n, z + 1, 1)
         Loop
           If v↓r ≠ 0 exit
           r := IF(r < n, r + 1, 1)
         z := r
       Liste := ADJOIN(v↓z, Liste)
       v := REPLACE(0, v, z)

Josef(10, 2) = [2, 4, 6, 8, 10, 3, 7, 1, 9, 5]
```

Now here is where our "indexfree programming" comes into play. As for the program logic let's first carry out the example above manually.

k	list	result of the action
0	[1,2,3,4,5,6,7,8,9,10]	initialization
1	[2,3,4,5,6,7,8,9,10,1]	cyclic rotation
2	[3,4,5,6,7,8,9,10,1]	truncation of 2
3	[4,5,6,7,8,9,10,1,3]	cyclic rotation
4	[5,6,7,8,9,10,1,3]	truncation of 4
5	[6,7,8,9,10,1,3,5]	cyclic rotation
6	[7,8,9,10,1,3,5]	truncation of 6
7	[8,9,10,1,3,5,7]	cyclic rotation
8	[9,10,1,3,5,7]	truncation of 8
9	[10,1,3,5,7,9]	cyclic rotation
10	[1,3,5,7,9]	truncation of 10
11	[3,5,7,9,1]	cyclic rotation
12	[5,7,9,1]	truncation of 3
13	[7,9,1,5]	cyclic rotation
14	[9,1,5]	truncation of 7
15	[1,5,9]	cyclic rotation
16	[5,9]	truncation of 1
17	[9,5]	cyclic rotation
18	[5]	truncation of 9
19	[5]	cyclic rotation
20	[ ]	truncation of 5

As you can see from the table above, we must alternate between cyclic rotations of the list and truncations of its first element. The elements, which are cut off, namely 2,4,6,8,10,3,7,1,9,5 must be collected in a list, which represents the Josephus permutation in the end.

As for an implementation of this idea, a first program could look like this:

```
myjosef0(n, s, k_ := 0, u_, v_ := []) :=
  Prog
    u_ := [1, ..., n]
    Loop
      k_ :+ 1
      If MOD(k_, s) = 0
         v_ := ADJOIN(FIRST(u_), v_)
         u_ := APPEND(u_, [FIRST(u_)])
      u_ := REST(u_)
      If u_ = []
         RETURN REVERSE(v_)
```

Well, this program is remarkably short, uses only 3 auxiliary variables and above all, it is free of indices! Nevertheless, its performance is still disappointing.

The reason is the line

```
u_ := APPEND(u_, [FIRST(u_)])
```

which performs the cyclic shifts. It is not at all a good idea, to do it this way. Remember, when dealing with a linked list it is relatively costly in terms of time to do things at the right end of a list. Hence, let's rewrite the program above in order to take this into account, even if it becomes slightly longer by this:

```
myjosef(n, s, k_ := 0, u_, v_ := [], w_ := []) :=
   Prog
     u_ := [1, ..., n]
     Loop
       k_ :+ 1
       If MOD(k_, s) = 0
          v_ := ADJOIN(FIRST(u_), v_)
          w_ := ADJOIN(FIRST(u_), w_)
       u_ := REST(u_)
       If u_ = []
          Prog
            If w_ = []
               RETURN REVERSE(v_)
            u_ := REVERSE(w_)
            w_ := []
```

As you can see, rather then appending the first elements at the end, we used an auxiliary list w_ and adjoined the corresponding elements at its beginning! What about the performance of this program? Does it make any difference? You bet! In order to test it, let's define another function interesting in its own:

```
survivor(n, s) := (myjosef(n, s))
                                  -1
```

Its purpose is to pick out the very last element of the Josephus permutation representing the survivor, as it were. (Well, referring to the original problem this lucky person could at least choose to commit suicide or simply stay alive!) For example, the computation

```
Survivor(10000,2)=3617
```

takes only 0.932s on my 2GHz-PC vs. 82.7s when using Rüdeger´s routine Josef().

Now I hear you saying that this is all very nice, but you expected to read more about number theory (or algebra for that matter) in this column and not so much about good programming techniques. Well, your wish is my command! Hence, let me conclude with a nice numbertheoretical application by deriving an explicit formula for the survivor function for the special case s=2, which we call f(n) in the following.

First, let's assume that n is even. Then in the first turn, i.e. going through the whole circle just once, all even numbers are wiped out and we are left with the odd numbers 1,3,5,7,…,n-1. Hence, assuming that we already know the survivor index f(n/2) for n/2 numbers a1,a2,…,an/2 it is easy to see that f(n)=2f(n/2)-1. Similarly, if n is odd, then after the first turn and the additional elimination of 1 in the second turn we are left with the odd numbers 3,5,7,…,n-1. By an analogous reasoning as above we see that f(n)=2f([n/2])+1 in this case, where [x] denotes the greatest integer $\leq$ x. Hence, we have already got the very efficient recursive definition

```
f(n):=if(n=1, 1, if(even?(n), 2f(n/2)-1, 2f(floor(n/2))+1))
```

The question remains whether this recursion can be used to get an explicit formula. To get a hint we use Derive to compute the first 30 values of f(n):

```
TABLE(f(n), n, 1, 30)'
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29

You see the pattern? If n is given in the form $n=2^i+j$ for some j with $0 \leq j < 2^i$, then $f(n)=f(2^i+j)=2j+1$. In other words, the explicit formula we are looking for is

```
f(n) := 2·(n - 2^FLOOR(LOG(n, 2)))+1
```

And here the final surprise as you should check yourself: Using the binary representation of n this is exactly a cyclic binary rotation of n to the left! (Any comments or suggestions, as always to j.wiesenbauer@tuwien.ac.at)

Calculation Time not only in the Status Line

René Hugelshofer, a close friend from Switzerland asked how to integrate calculation time into the output of a function and/or program in Derive – in addition to the computation time presented in the Derive status line. It is a shame that I didn´t have an answer at the moment. I met Albert Rich in Montréal and this was the occasion to ask the right person. He answered that all what I want to know can be found in the Online Help, and he is right:

`RANDOM(0)` simplifies to the time in centiseconds since the current calendar year began and initializes the random number state variable to that time.

$$a := \begin{bmatrix} x + y & x^2 & y^2 & z \\ z + 4 & x + 3 & y - 1 & x \cdot z \\ y^2 & x^4 & z^2 & x - z \\ x & y & z & x + y + z \end{bmatrix}$$

$$a^{-1}$$

This is a time consuming operation and we would like to include the computation time into the output.

Simplifying the inverse matrix gives a "nice" expression and takes 3.06 sec (status line) on my PC. Time differs a little from one calculation to the other.

The following short program does the job (I am sure that you can follow the code):

```
      inv_time(x, t_, x_, time) :=
        Prog
          t_ := RANDOM(0)
#4:       x_ := x^(-1)
          time := APPROX((RANDOM(0) - t_)/100)
          RETURN [x_, APPEND("CompTime = ", STRING(time))]

#5:   inv_time(a)
```

The status line reports a computation time of 3.55 sec. What is the reason for the the difference?
The answer is seems to be easy:
Formatting this bulky expression might take some time. Let´s test this by using another program avoiding the output.

$$\left| \frac{}{2 \quad 2} \right.$$
$$\cdot(z^2 + 7 \cdot z + 3))$$
$$\left| \frac{}{2 \quad 2} \right.$$

, CompTime = 2.85

```
      inv_time2(x, t_, time) :=
        Prog
          t_ := RANDOM(0)
#6:       x_ := x^(-1)
          time := APPROX((RANDOM(0) - t_)/100)
          RETURN APPEND("Result in x_, CompTime = ", STRING(time))

#7:   inv_time2(a)

#8:   Result in x_, CompTime = 2.64
```

Status line shows 2.69 sec. Calling now the result by simplifying **x_** as next expression #9, the status line shows 0.38 seconds, which makes the difference. Please note, that **x_** is global in this case!

Many thanks René for this interesting question.

Amusing Corner of the Deriver´s (and other CASer´s)-Curiosity

founded by Alfonso Población Saéz

Hello Josef and Noor,

It was nice seeing you in Montreal. Perhaps you have heard of the following puzzle.

In July, billboards and banners began appearing with the cryptic message:

{ First 10 digit prime in consecutive digits of e }.com

If you solved this puzzle, you were led to another puzzle, and ultimately to Google's webpage. These puzzles were a ploy to recruit more engineers to work for Google.

I did not hear about this problem until about one week ago. Of course, hundreds of other people have already solved this prime puzzle. Still, it seemed like a good problem for DERIVE – and it was. After finding the first 10-digit prime in consecutive (decimal) digits of e, I decided to find the second, third, etc. In fact, I found all the 10-digit primes contained in the first 1000 decimal digits of e. I wrote a DERIVE procedure (program) to find all the n-digit primes in the first 1000 digits of e; for a value of n = 1, 2, etc. determined by the user. The ***1000 digits*** was just a convenient, arbitrary, fairly large number. I could have searched the first 1093 digits of e.

Let me challenge your readers to write similar procedures. Of course, they might want to search other irrational numbers {PI, SQRT(2), SQRT(3), ATAN(2), etc.} for n-digit primes.

Remarks:

(1) Searching the first 1000 digits of a rational number should give routine, predictable results. Hence, you may want to search for primes exclusively in irrational numbers.

(2) Of the 10 one-digit numbers, {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, there are exactly four primes {2, 3, 5, 7}. So, you might expect to find approximately 40% of the one-digit numbers in e to be primes. This percentage will decrease when you search for longer primes. Try to find a relationship between the length of these primes and the number found in the first 1000 digits of e.

(3) In stead of searching for 10-digit primes contained entirely in the first 1000 digits of e, you might want to search for these primes with leading (first) digit among the first 1000 digits of e. This way, you will always search through the same number of candidates (1000) – whether you search for 10-digit primes or 20-digit primes.

(4) In searching for 10-digit primes, my program will return:

(a) a 9-digit prime – if it is preceded by one zero digit;

(b) an 8-digit prime – if it is preceded by two zero digits;

(c) a 7-digit prime – if it is preceded by three zero digits; etc.

These "shorter" primes can easily be sorted out with a second search. Also, you may want to determine what percent of the primes are "shorter" primes.

(5) As of this writing, you can still check your first 10-digit prime in e by concatenating

.com

and using it as a www address. If correct, you will get to the second Google puzzle.

Happy Searching,

Steven Schonefeld